Non-Black-Box Techniques in Cryptography

Thesis for the Ph.D. Degree

by

Boaz Barak

Under the Supervision of
Professor Oded Goldreich

Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science

Submitted to the Feinberg Graduate School of
the Weizmann Institute of Science
Rehovot 76100, Israel

January 5, 2004

Abstract

The American Heritage dictionary defines the term "Black-Box" as

> *"A device or theoretical construct with known or specified performance characteristics but unknown or unspecified constituents and means of operation."*

In the context of Computer Science, to use a program as a *black-box* means to use only its input/output relation by executing the program on chosen inputs, without examining the actual code (i.e., representation as a sequence of symbols) of the program.

Since learning properties of a program from its code is a notoriously hard problem, in most cases both in applied and theoretical computer science, only black-box techniques are used. In fact, there are specific cases in which it has been either proved (e.g., the HALTING Problem) or is widely conjectured (e.g., the SATISFIABILITY Problem) that there is no advantage for non-black-box techniques over black-box techniques.

In this thesis, we consider several settings in cryptography, and ask whether there actually is an advantage in using *non-black-box* techniques over *black-box* techniques in these settings. Somewhat surprisingly, our answer is mainly positive. That is, we show that in several contexts in cryptography, there *is* a difference between the power of black-box and non-black-box techniques. Using non-black-box techniques we are able to solve some problems in cryptography that were previously unsolved. In fact, some of these problems were previously proven to be unsolvable using black-box techniques.

The main results of this thesis are the following:

Software Obfuscation Informally speaking, an *obfuscator* is a compiler that takes a program P as input and produces a new program P' that has the same functionality as P, and yet is "unintelligible" in some sense. Ideally, a software obfuscator should ensure that the only information leaked about P from the program P', is information that can be derived by using only black-box access to P. Obfuscators, if they exist, would have a wide variety of cryptographic and complexity-theoretic applications, ranging from software protection to homomorphic encryption to complexity-theoretic analogues of Rice's theorem.

In this thesis, we discuss how to formally define obfuscators, and whether or not such objects exist. Our main result in this context is that even very weak forms of obfuscators do not exist.

Zero-Knowledge The simulation paradigm, introduced by Goldwasser, Micali and Rackoff, has had fundamental impact on cryptography. A *simulator* is an algorithm that tries to simulate the interaction of the adversary with an honest party, without knowing the private input of this honest party. Loosely speaking, the existence of such a simulator demonstrates that the adversary did not gain any knowledge about the honest party's input.

Almost all previously known simulators used the adversary's algorithm as a black-box. We present the first constructions of *non-black-box* simulators. Using these new non-black-box techniques we obtain several results that were previously shown to be impossible to obtain using black-box simulators.

Specifically, assuming the existence of collision-resistant hash functions, we construct a new constant-round zero-knowledge argument system for NP that satisfies the following properties:

1. It remains zero knowledge even when composed concurrently n times, where n is the security parameter.
2. It is an Arthur-Merlin (public coins) protocol.
3. It has a simulator that runs in *strict* probabilistic polynomial-time, rather than in *expected* probabilistic polynomial-time.

It is impossible to obtain a constant-round zero-knowledge argument (for a non-trivial language) satisfying either Property 1 or Property 2 using black-box simulation (Canetti et al., 2001), (Goldreich and Krawczyk, 1996). We show that it is also impossible to obtain a constant-round zero-knowledge argument satisfying Property 3 using black-box simulation.

We use this protocol to obtain other new results in cryptography. These include a construction of constant-round zero-knowledge proof of knowledge with a strict polynomial-time knowledge extractor, a zero-knowledge resettably sound argument for **NP**, and a resettable zero-knowledge argument of knowledge for **NP**. We show that all these applications are impossible to obtain when restricted to black-box techniques.

Non-Malleability We construct the first constant round non-malleable commitment scheme and the first constant-round non-malleable zero-knowledge argument system, as defined by Dolev, Dwork and Naor (Siam J. Computing, 2000). Previous constructions either used a non-constant number of rounds, or were only secure under stronger setup assumptions (such as the availability of a public string that was chosen at random and published by a trusted third party).

We obtain this result by using a *non-black-box* proof of security. That is, our proof uses the code of the adversary in the security reduction.

As an intermediate step we define and construct a constant-round non-malleable coin tossing protocol. This coin-tossing protocol may be of independent interest.

To summarize, in this thesis we show that, somewhat unintuitively, non-black-box techniques sometimes have a significant advantage over black-box techniques in cryptography. From the point of view of cryptographers, this result has both negative and positive applications. On the one hand, it further stresses the point that it is unsafe to rely on the assumption that an adversary attacking our schemes will use only black-box techniques. On the other hand, it means that when designing and analyzing cryptographic schemes, we can use these non-black-box techniques to obtain useful and important security goals that cannot be obtained using black-box techniques.

Origin of the term "Black-Box"

The term "black-box" has two common interpretations. One is a flight recorder on an airplane, that records all information regarding the flight, and helps discover the reasons for a crash.[1] Another interpretation, which is the one we are interested in, is *"A device or theoretical construct with known or specified performance characteristics but unknown or unspecified constituents and means of operation"* [Ame00].

It seems that both interpretations have the same origin. In the second world war, "black-box" was a slang word among the pilots of the British Royal Air-Force (RAF) for radar instruments [Par90, Has03]. One reason was simply that these instruments were indeed enclosed in black boxes. There are also sources that suggest that the pilots, who didn't know how these instruments operated (and in fact were not allowed to know this, since it was classified information) joked that they operated on "black magic".[2] Indeed, this is probably the reason this term came to describe any piece of mysterious electronics, which its users know (or at least have some clue as to) *what* it does, but have no idea *how* it does it.

Although all sources seem to agree that the term originated from the RAF, there are earlier examples of mysterious devices being placed in black boxes (c.f. [Vas96]). Perhaps the most notable of these was the *Oscilloclast* (also called the ERA), invented by Dr. Albert Abrams in the beginning of the 20^{th} century (see [Sch01, Edw00, Vle99, Abr16]). The Oscilloclast was a black box that was supposed to cure all sorts of various diseases. 3500 practitioners were using the Oscilloclast at the height of its popularity in 1923. As in the case of modern software "black boxes", the Oscilloclast, which was leased for $200 down and $5 per month, was sealed and the lessee had to sign a contract not to open it. Thousands of patients were diagnosed and cured by Abrams of bovine syphilis, a disease whose existence was never established to the satisfaction of the medical profession. In addition, when a blood sample from a healthy guinea-pig was sent to Abrams, it was diagnosed as suffering general cancer and tuberculosis of the genito-urinary tract, another diagnosis of a drop of sheeps blood came back as hereditary syphilis with an offer of a cure for $250. When physicists and engineers opened the devices they found them to be essentially a jungle of electric wires.

Thanks to `justaskscott-ga` from the Google Answers website.

[1]Today these recorders are usually painted in a bright orange color.

[2]For example consider this quote from an RAF fighter pilot John Cunnigham expressing his frustration with the early radar systems: *"The magician was still kneeling on his prayer mat of blankets muttering to himself, the green glow from the CRT flickered on his face. A witch doctor, I thought, a witch doctor and black magic – and just about as useful."*. See `http://www.vectorsite.net/ttwiz2.html`.

Acknowledgements

First and foremost, I want to thank my advisor, Oded Goldreich. Although it was certainly not always easy, I think that I have evolved in the last four years, both as a scientist and as a person. Oded, more than any other person, was instrumental in this evolution. It was mainly through interaction with Oded that I shaped my understanding of my subject, of Computer Science at large, and of being a scientist. (Although, of course, any errors in the end result are my responsibility...) I feel very lucky to have had Oded as an advisor, and look forward to having more opportunities to learn from and work with him in the future.

One of the many reasons I am grateful to Oded was his encouragement and support of my meeting and working with other scientists. As a result, in a relatively short time, I have met, interacted and worked with so many wonderful people, that I cannot mention all of them here. However, I do want to mention two scientists Oded introduced me to, that meeting them has had a particularly strong impact on me. Salil Vadhan was the first person I cooperated with other than Oded, and since then I had the chance to work with Salil on several occasions. It is hard to pinpoint exactly what makes talking with Salil so enjoyable, but I think that his ability to understand everything I say (even when I don't really understand it myself), has something to do with it. Avi Wigderson hosted me during the summer of 2001 in the Institute for Advanced Study in Princeton. I discovered there that it is nearly impossible to spend five minutes with Avi without learning something new. Needless to say, I am looking forward to spending more time with him in the (very near) future.

I would also like to thank Tal Rabin and all the members of the cryptography research group in the IBM T.J.Watson research center – Ran Canetti, Rosario Gennaro, Shai Halevi, Hugo Krawczyk and Yehuda Lindell – for hosting me during a fun and productive summer visit. The fact that Tal is one of the nicest and most fun to be with persons I have ever met certainly didn't hurt.

Like all the students I know, I am very happy I've had the chance to study Computer Science in the Weizmann Institute. Our department has a unique atmosphere of academic excellence combined with friendliness and openness that makes it a very special place to learn in. I greatly enjoyed the interaction with quite a lot faculty members, postdocs and fellow students. This includes people I wrote papers with such as Shafi Goldwasser, Yehuda Lindell, Ronen Shaltiel and Eran Tromer, and people I "just" talked with such as Itai Binyamini, Uri Feige, Moni Naor, Ran Raz, Alon Rosen, Adi Shamir and Amir Shpilka. I would also like to thank Shafi and Adi for serving on my thesis committee. My time in the institute would have been much duller without my office and cubicle mates Adi Akavia, Eran Ofek, Itsik Mantin, Eran Tromer, and (especially) Udi Wieder – thank you!

Finally, I would like to thank my family, and especially my parents Dov and Judith, and my sweet wife Ravit. They made completing this project possible and worthwhile.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A *computer program* (or equivalently, an *algorithm*) is a list of symbols – a finite string. When we interpret a string Π as a program, we associate with this string a function that the string Π *computes*. For example, if we interpret the string

```
int f(int x) {
    return x+1;
}
```

as a program in the C programming language, then we associate with it the function $f(\cdot)$ where $f(x) \stackrel{def}{=} x + 1$ for any integer x. Given a program Π and a value x, it is possible to compute $f(x)$, where $f(\cdot)$ is the function associated with the program Π, by executing the program Π on a computer. We sometimes call the string Π the *representation* or *code* of the function $f(\cdot)$.

Since a program is a string, it sometimes makes sense to use it as input to a different program. Indeed, algorithms that take other programs as input are very common in Computer Science. In most cases, these algorithms use their input program as a *subroutine*. By this we mean that, on input a program Π, the algorithm's operation does not depend on the particular representation of the program Π, but rather, the algorithm only uses its input program Π to evaluate the *function* that the program Π computes. That is, the algorithm only uses Π to obtain a "black box" for computing $f(\cdot)$, such that the algorithm can feed inputs to and receive outputs from this box. We call such an algorithm (that only uses black-box access to the program it gets as input) a *black-box* algorithm.

As mentioned above, black-box algorithms are very popular in computer science. Many times, when trying to solve a particular task, one would write an algorithm that solves the task if it is given black-box access to programs that solve some simpler tasks, and then write programs that solve these simpler tasks. This approach is a basic paradigm of software engineering, and almost all programming languages implement mechanisms (such as function calls) to facilitate it.

Black-box algorithms are also very popular in the more theoretical aspects of Computer Science. For example, when proving that a decision problem L is **NP**-complete, one needs to show the existence of a black-box algorithm B (where B is usually called a *reduction*), such that if B is given black-box access to an algorithm A that solves the problem L, then B can solve the SATISFIABILITY problem. Such reductions also appear in cryptography. For example, consider the constructions of public key encryption schemes whose security can be reduced to the factoring problem [Rab79, BG84]. This is shown by providing a black-box algorithm B, such that if B is given black-box access to an algorithm A that compromises the security of the encryption scheme, then B can solve the factoring problem.

Why are black-box algorithms so popular? The reason that black-box algorithms are so popular is that it seems very hard to make use of the particular representation of a program as a string. Understanding the properties of a function from the code of a program that computes it (also known as *reverse-engineering*) is a notoriously hard problem. In fact, considerable efforts are made at writing programs so they would be easy to understand. Programs written without such efforts, or programs written or compiled to low level languages are considered to be quite incomprehensible. In fact, in some cases it is either *proven* (e.g., the HALTING problem, Rice's theorem) or widely conjectured (e.g., the SATISFIABILITY problem) that when trying to learn properties of a function, there is no significant advantage to getting its representation as a program, over getting black-box access to it. Thus, a common intuition is the following:

The only useful thing one can do with a program is to execute it (on chosen inputs).

In this thesis, we test this intuition in several settings in cryptography. Somewhat surprisingly, we find several cases in which it does *not* hold. That is, we find several cases in which it can be shown that *non-black-box* algorithms have significantly more power than *black-box* algorithms. We use this additional power of non-black box algorithms to obtain new results. Some of these results were previously proven to be *impossible* to obtain when using only black-box techniques.

1.1 Our Results

The main theme of this thesis is that non-black-box techniques *can* indeed be more powerful than black-box techniques in several interesting contexts in cryptography. Below, we elaborate more on our specific results in particular contexts. We only mention here our results, and we do not elaborate on the ideas and techniques used to obtain these results. Each chapter in this thesis has its own introduction, which contains a much more detailed (but still high level) discussion on the results of the chapter and on the ideas used in the proofs. We note that we present the results here in a different order than the order of the chapters. This is because the results of Chapter 6 are somewhat easier to state than the results of Chapters 4 and 5, but the proofs of Chapter 6 are actually more complicated and use some ideas from the previous chapters.

1.1.1 Code Obfuscation

Code obfuscation is about trying to make practical use of the difficulty of reverse-engineering programs. Informally, an obfuscator $\mathcal{O}$ is an (efficient, probabilistic) "compiler" that takes as input a program Π and produces a new program $\mathcal{O}(\Pi)$ that has the same functionality as Π yet is "unintelligible" in some sense. Obfuscators, if they exist, would have a wide variety of cryptographic and complexity-theoretic applications, ranging from software protection to homomorphic encryption to complexity-theoretic analogues of Rice's theorem. Most of these applications are based on an interpretation of the "unintelligibility" condition in obfuscation as meaning that $\mathcal{O}(\Pi)$ is a "virtual black box," in the sense that anything one can efficiently compute given $\mathcal{O}(\Pi)$, one could also efficiently compute given black-box access to Π.

Several constructions of software obfuscators have been previously suggested (c.f., [CTL97] and the references therein). However, no formal definition of obfuscation has been suggested, and so in particular none of these candidates has been proven to meet some formal security definition. In Chapter 3 of this thesis, we initiate a theoretical investigation of obfuscation. Our main result is that, even under very weak formalizations, obfuscation is *impossible*. We prove this by constructing a family of programs $\mathcal{P}$ that are *inherently unobfuscatable* in the sense that

1. There is an efficient algorithm A such that for every program $\Pi \in \mathcal{P}$ and every program Π' that computes the same function as Π, $A(\Pi') = \Pi$. That is, for every possible obfuscator $\mathcal{O}$, A can recover the original source code of Π from $\mathcal{O}(\Pi)$.

   but

2. When given only *black box* access to a program Π chosen at random from $\mathcal{P}$, it is infeasible to compute the program Π.

We extend our impossibility result in a number of ways, including even obfuscators that **(a)** are not necessarily computable in polynomial time, **(b)** only approximately preserve the functionality, and **(c)** only need to work for very restricted models of computation ($\mathbf{TC_0}$). We also rule out several potential applications of obfuscators, by constructing "unobfuscatable" signature schemes, encryption schemes, and pseudorandom function families.

1.1.2 Non-black-box Proofs of Security

A typical cryptographic theorem has the following form "Scheme X (e.g., a secure voting protocol) is as secure as Problem Y (e.g., factoring random Blum integers)". This statement means that if there exists an efficient algorithm A that breaks the security of the scheme X, then there exists an efficient algorithm B that can solve the problem Y. In all previous cases that we are aware of, such statements were proven via *black-box reductions*. That is, to show that the statement is true, one gave a construction of a generic algorithm B that takes as input both an instance of the problem Y and uses *black-box* access to an algorithm A. Then, one proves that if A is an algorithm to break the scheme X, then when given access to A, Algorithm B solves the problem Y.

A natural question is whether one can gain more power by using a *non-black-box* reduction. That is, whether by considering also reductions that let the algorithm B use also the *code* of A one can obtain new cryptographic schemes. In Chapter 6 we give a positive answer to this question. We use there a *non-black-box* proof of security to construct the first constant round non-malleable commitment scheme and the first constant-round non-malleable zero-knowledge argument system, as defined by Dolev, Dwork and Naor [DDN91].

Non-malleability is a strengthened notion of security that is needed for some applications of secure protocols. In particular, non-malleable commitment schemes capture better the intuitive notion of "digital envelopes" in the sense that the committed value is not only hidden from the receiver of such a commitment scheme but also the receiver cannot form a commitment to any related value. In contrast, when using a standard (i.e., malleable) commitment scheme, it may be the case that the receiver of a commitment to a value x, can form a commitment to a value related to x (e.g., $x + 1$) even though he cannot learn x.

Previous constructions of non-malleable commitment schemes and zero-knowledge proofs either used a non-constant number of rounds, or were only secure under stronger setup assumptions (such as the availability of a public string that is chosen at random and published by a trusted third party).

As an intermediate step we define and construct a constant-round non-malleable *coin tossing* protocol. This coin-tossing protocol may be of independent interest.

1.1.3 Non-Black-Box Simulation

The *simulation paradigm*, introduced by Goldwasser, Micali, and Rackoff [GMR85], is one of the most important paradigms in the definition and design of cryptographic primitives. For example, this paradigm arises in a setting in which two parties, Alice and Bob, interact in some secure protocol (e.g., a zero-knowledge proof) and Bob knows a secret. We want to make sure that Alice hasn't learned anything about Bob's secret as the result of this interaction, and do so by showing that Alice could have *simulated the entire interaction by herself*. Therefore, she has gained no further knowledge as the result of interacting with Bob, beyond what she could have discovered by herself.

Formally, this is shown by exhibiting a *simulator*. A simulator is an algorithm, that gets as input an algorithm A^* which describes Alice's strategy in the protocol, and outputs a distribution that is indistinguishable from the distribution of the messages that Alice sees in a real interaction with Bob when she is using the strategy A^*. The existence of such a simulator demonstrates that regardless of the strategy A^* that Alice's uses, she has not learned anything about Bob's secret that she couldn't have learned by herself without having any interaction with Bob (by simply running the simulator).

Almost all previously known simulators used only *black-box* access to the algorithm A^* they received as input.[1] In Chapter 4, we present the first construction of a protocol with a *non-black-box* simulator under standard assumptions. Using these new non-black-box techniques we obtain several results that were previously shown to be impossible to obtain using black-box simulators.

Specifically, assuming the existence of collision-resistent hash functions, we construct a new zero-knowledge argument (i.e., a computationally-sound proof) for any language in **NP** that satisfies the following properties:

1. It is zero-knowledge with respect to non-uniform adversaries with auxiliary information.

2. It has a constant number of rounds and negligible soundness error.

3. It remains zero-knowledge even if executed concurrently n times, where n is the security parameter. We call a protocol that satisfies this property a *bounded concurrent zero-knowledge* protocol.[2]

4. It is an Arthur-Merlin (public coins) protocol.

5. It has a simulator that runs in *strict* probabilistic polynomial-time, rather than *expected* probabilistic polynomial-time.

The above protocol should be contrasted with the following impossibility results regarding *black-box* zero-knowledge arguments for non-trivial languages: Goldreich and Krawczyk [GK90] showed that such protocols cannot satisfy both Properties 2 and 4. Canetti, Kilian, Petrank and Rosen [CKPR01] showed that such protocols cannot satisfy both Properties 2 and 3. In Chapter 4, we also show that such protocols cannot satisfy Properties 2 and 5.

In addition, in Chapter 5 we use this zero-knowledge system to obtain other new results in cryptography. These applications include **(a)** a construction of constant-round zero-knowledge argument of knowledge with a strict polynomial-time knowledge extractor, **(b)** a zero-knowledge resettably sound argument for NP, and **(c)** a resettable zero-knowledge argument of *knowledge*. We show that all these three applications are impossible to obtain when restricted to black-box techniques.

[1] One exception is the zero-knowledge proof system of [HT99]. However, that protocol was constructed under a computational assumption that the authors themselves describe as unreasonable.

[2] The choice of n repetitions is quite arbitrary and could be replaced by any *fixed* polynomial (e.g. n^3) in the security parameter. This is in contrast to a standard concurrent zero-knowledge protocol [DNS98, RK99] that remains zero-knowledge when executed concurrently any polynomial number of times.

We remark that application **(b)** in particular, is somewhat counter-intuitive, and demonstrates well the power of non-black-box techniques. In particular it means that if you are given a device (e.g., a smart-card) that proves some statement σ in this system then you are not able to learn anything new about σ except its validity by "playing" with the device – feeding it with different inputs, and examining its outputs. However you can still be certain that if you were to open the device and examine its internal workings, you would be able to extract a witness for σ.

Relation to non-black-box proofs of security. We remark that in some sense non-black-box simulation is a special case of non-black-box proofs of security. This is because one can view the existence of a simulator as a proof that the protocol is secure, and so a non-black-box simulator can be viewed as a non-black-box proof of security. Indeed, usually when a zero-knowledge protocol is used as a component in a larger sub-protocol, the security reduction for the larger protocol involves using the simulator for the zero-knowledge protocol. Thus, if the zero-knowledge protocol used has a non-black-box simulator, then the security reduction for the larger protocols will be non-black-box.

1.2 How to read this thesis.

One way to read this thesis is of course to read it sequentially, from cover to cover. Another way to read it is to start by reading the introductions to Chapters 3–6, and then continue to the actual contents. The introduction to each chapter contains also an organization section that specifies the dependency structure of the individual sections within the chapter.

Dependency between chapters. The following information might be useful to readers that want to read only individual chapters.

- Chapter 2 contains notations and definitions used throughout the thesis. Most readers can probably read just the summary of this chapter, and return to specific definitions in it when necessary.
- Chapter 5 depends on the results proven in Chapter 4 and thus should be read after Chapter 4.
- Although, technically Chapter 6 does not depend on the results of the previous chapters, it uses similar ideas to Chapter 4 in a more complicated setting. Thus I recommend reading Chapter 4 before Chapter 6.
- Universal arguments are used both in Chapter 4 and Chapter 6. Their definition is presented in Chapter 2 and a construction is presented in Appendix A.

Chapter 2

Preliminaries and Definitions

Summary: This chapter contains the notations and basic definitions used throughout this thesis. The reader may want to skim through this chapter, since in most cases we follow the standard notations and definitions used in the literature (e.g., in [Gol01b, Gol04]). Notable exceptions are the following:

1. We use the notion of a *random-access hashing scheme* (also known as tree hashing or Merkle's hash trees [Mer89]). Although similar notions have been used before, the name and definition we use are somewhat different.
2. We use the notion of *universal arguments*, which are a cross between CS proofs and argument systems. Our definition is taken from [BG01].
3. We use the notion of a *commit-with-extract* scheme. Similar concepts have been used in the literature, but our definition is taken from [BL02].
4. We take a unified approach for defining zero knowledge and proofs of knowledge. In both cases we define them as having a *non-black-box universal* simulator/extractor. Most previous works defined zero-knowledge as having a non-black-box non-universal simulator, and proofs of knowledge as having a black-box universal knowledge extractor.

We refer the reader to Goldreich's textbooks ([Gol01b, Gol04]) for a much more extensive treatment of the concepts mentioned in this section, including discussions on the motivations and context of these concepts.

2.1 Standard Notations

Functions. We use standard mathematical notations regarding functions. That is, if f is a function that maps elements of a set X to elements of a set Y, then we denote this by $f : X \to Y$. We say that X is the *domain* of f and Y is the *range* of f. If $A \subseteq X$ then $f(A)$ denotes the set $\{f(a) \mid a \in A\} = \cup_{a \in A}\{f(a)\}$. If $y \in Y$ then $f^{-1}(y)$ denotes the set $\{x \in X \mid f(x) = y\}$. If $B \subseteq Y$ then $f^{-1}(B)$ denotes the set $\cup_{b \in B} f^{-1}(b)$. We also use the standard O-notations $(O, \Omega, o, \omega, \Theta)$ to denote order of growth of functions.

Distributions. For a finite set $S \subseteq \{0,1\}^*$, we write $x \leftarrow_{\mathrm{R}} S$ to say that x is distributed uniformly over the set S. If D is a distribution, then we write $x \leftarrow_{\mathrm{R}} D$ to say that x is a random variable distributed according to D. We use $\mathrm{Supp}(D)$ to denote the *support* of a distribution D (i.e., the set of all elements with non-zero probability). We denote by U_n the uniform distribution over the set $\{0,1\}^n$. A function $\mu(\cdot)$, where $\mu : \mathbb{N} \to [0,1]$ is called *negligible* if $\mu(n) = n^{-\omega(1)}$ (i.e., $\mu(n) < \frac{1}{p(n)}$ for all polynomials $p(\cdot)$ and large enough n's). We say that an event happens with *overwhelming* probability if it happens with probability $1 - \mu(n)$ for some negligible function $\mu(\cdot)$. We will sometimes use neg to denote an unspecified negligible function. If α and β are strings, then $\alpha \circ \beta$ denotes the *concatenation* of α and β.

Computational models, encoding algorithms as strings. We assume the standard notions of Turing machines and Boolean circuits. For all of the results of this thesis, details such as the number of tapes of Turing machine, or the types of gates allowed in a Boolean circuit (as long as they generate all Boolean functions), will not matter. We identify algorithms with their description as either Turing machines or Boolean circuits in some canonical way (for the purposes of this thesis, it does not matter which reasonable method one chooses to encode Turing machines or Boolean circuits as strings). For example, $A(A)$ denotes the output of the algorithm A, when given as input the string that describes A. In some cases, for added clarity, we may denote the description of an algorithm A by $\mathsf{desc}(A)$ (and thus, in the example above, one can denote the output of A when given as input the description of A also by $A(\mathsf{desc}(A))$). The *size* of an algorithm A is the length of the string that describes it. A polynomial-sized circuit family is a sequence of circuit $\{C_n\}_{n\in\mathbb{N}}$ such that each circuit C_n is of size polynomial in n, has $n^{\Theta(1)}$ input bits, and $n^{O(1)}$ output bits. We also use the standard notions of *oracle Turing machines* and circuits with oracle gates. We denote the execution of an algorithm A on input x with access to oracle O by $A^O(x)$. We will sometimes use $[A]$ to denote the (possibly partial) function that a an algorithm A computes.

Black-box subroutines. In many cases we will invoke an algorithm A with the description of an algorithm B as part of its input. In such cases we say that A *uses* B *as a black-box subroutine* if the only use A makes of the description B is to execute B on chosen inputs (i.e., A is essentially an oracle algorithm that makes oracle calls to B). In such cases we may say that A is a *black-box algorithm.*

Languages and witnesses. Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. We define $L(R) \stackrel{def}{=} \{x \mid \exists y \text{ s.t. } (x,y) \in R\}$. If $x \in L(R)$ and y is a string such that $(x,y) \in R$ then we say that y is a *witness* for the fact that $x \in L(R)$. We denote the set of witnesses to x by $R(x)$. That is, $R(x) = \{y \mid (x,y) \in R\}$. Let $T : \mathbb{N} \to \mathbb{N}$ be some function. We say that $L \in \mathbf{Ntime}(T(n))$ if there exists a relation R such that $L = L(R)$ and a Turing machine M such that on input (x,y), the machine M runs for at most $T(|x|)$ steps and output 1 if $(x,y) \in R$ and 0 otherwise.

Probabilistic algorithms. If A is a probabilistic algorithm, then we let $A(x;r)$ denote the output of A on input x and random-tape r. We let $A(x)$ denote the random variable that represents $A(x;r)$ for a randomly chosen r of appropriate length.

Adversarial models. Our standard way to model an efficient adversary strategy will be a family of polynomial-sized circuits. However, we also consider other models such as $T(n)$-sized circuits for a *super-polynomial* function $T : \mathbb{N} \to \mathbb{N}$ (e.g., $T(n) = n^{\log n}$ or $T(n) = 2^{n^{1/10}}$). We will also sometimes consider *uniform* adversaries. That is, adversaries that are described using probabilistic polynomial-time Turing machines. In fact, in some cases, we will also consider a “hybrid model” of adversaries with *bounded non-uniformity.* Such adversaries are described by a probabilistic polynomial-time Turing machine that on inputs of size n gets an advice string of length $l(n)$ where $l : \mathbb{N} \to \mathbb{N}$ is some fixed function that is polynomially related to n. We stress that the running time of such adversaries may be any polynomial and so in particular may be larger than $l(n)$.

Note: In most cryptographic works, a proof of security for *uniform* adversaries, can be extended to yield a proof of security for *non-uniform* adversaries (under appropriate complexity assumptions). However, this is only because these works use *black-box* reductions in their proof of security. In contrast, in this thesis we will often utilize *non-black-box* techniques. This means that our proofs of security against uniform adversaries will often *not* extend *automatically* into proofs of security against non-uniform adversaries. Indeed, we will often need to use more complicated proofs and additional ideas to extends proofs from the uniform into the non-uniform setting.

Computational indistinguishability. Let X and Y be random variables over $\{0,1\}^n$ and let $s \geq n$. We say that X and Y are *indistinguishable by s-sized circuits* if for every circuit D of size s, it holds that $|\Pr[D(X) = 1] - |\Pr[D(Y) = 1]| < \frac{1}{s}$. A *probability ensemble* is a sequence $\{X_i\}_{i \in I}$ of random variables, where I is an infinite subset of $\{0,1\}^*$ and X_i ranges over $\{0,1\}^{p(|i|)}$ for some polynomial $p(\cdot)$. We say that two probability ensembles $\{X_i\}_{i \in I}$ and $\{Y_i\}_{i \in I}$ are *computationally indistinguishable*, denoted by $\{X_i\}_{i \in I} \equiv_{\mathrm{C}} \{Y_i\}_{i \in I}$, if for every polynomial $p(\cdot)$ and every sufficiently large i, X_i and Y_i are indistinguishable by $p(|i|)$-sized circuits. An equivalent formulation is that $\{X_i\}_{i \in I}$ and $\{Y_i\}_{i \in I}$ are computationally indistinguishable if there exists a negligible function $\mu : \mathbb{N} \to [0,1]$ such that X_i and Y_i are indistinguishable by $\frac{1}{\mu(|i|)}$-sized circuits. We will sometimes abuse notation and say that the two random variables X_i and Y_i are computationally indistinguishable, denoted by $X_i \equiv_{\mathrm{C}} Y_i$, when each of them is a part of a probability ensemble such that these ensembles $\{X_i\}_{i \in I}$ and $\{Y_i\}_{i \in I}$ are computationally indistinguishable. We will also sometimes drop the index i from a random variable if it can be inferred from the context. In most of these cases, the index i will be of the form 1^n where n is called the *security parameter.* We will use the following basic facts regarding computational indistinguishability:

Proposition 2.1.1. *Let M be a probabilistic polynomial-time Turing machine. If $\{X_i\}_{i \in I}$ and $\{Y_i\}_{i \in I}$ are computationally indistinguishable then so are $\{M(X_i)\}_{i \in I}$ and $\{M(Y_i)\}_{i \in I}$.*

We call a probability ensemble $\{X_i\}_{i \in I}$ *efficiently sampleable* if there exists a probabilistic polynomial-time Turing machine S and a polynomial $p(\cdot)$ such that $X_i = S(i, U_{p(|i|)})$. We have the following two facts about efficiently sampleable ensembles:

Proposition 2.1.2. *Let $\{X_i\}_{i \in I}$, $\{Y_i\}_{i \in I}$, $\{U_i\}_{i \in I}$ and $\{T_i\}_{i \in I}$ be four efficiently sampleable probability ensembles. If $\{X_i\}_{i \in I}$ is computaionally indistinguishable from $\{Y_i\}_{i \in I}$ and $\{U_i\}_{i \in I}$ is computationally indistinguishable from $\{T_i\}_{i \in I}$ then the ensemble $\{(X_i, U_i)\}_{i \in I}$ is computationally indistinguishable from the ensemble $\{(Y_i, T_i)\}_{i \in I}$, where (X_i, U_i) (resp. (Y_i, T_i)) represents a pair (x, u) (resp. (y, t)) such that x (resp. y) is sampled from X_i (resp. Y_i) and u (resp. t) is independently sampled from U_i (resp. T_i).*

Proposition 2.1.3. *Let $\{X_i\}_{i \in I}$ and $\{Y_i\}_{i \in I}$ be two efficiently sampleable and computationally indistinguishable probability ensembles. Let $p(\cdot)$ be some polynomial. Then, the ensembles $\{(X_i^{(1)}, \ldots, X_i^{(p(|i|))})\}_{i \in I}$ and $\{(Y_i^{(1)}, \ldots, Y_i^{(p(|i|))})\}_{i \in I}$ are computationally indistinguishable, where $(X_i^{(1)}, \ldots, X_i^{(p(|i|))})$ (resp. $(Y_i^{(1)}, \ldots, Y_i^{(p(|i|))})$) represents $p(|i|)$ independent copies of X_i (resp. Y_i).*

2.2 Basic Cryptographic Primitives

In this section we define some basic cryptographic primitives. We define all of these to be secure against polynomial-sized circuits, but analogous definitions can be obtained for other adversary models. In particular, we will sometimes use variant of these definitions where security should hold against $T(n)$-sized circuits, where $T(n)$ is some fixed super-polynomial function (e.g., $T(n) = 2^{n^\epsilon}$ for some constant $\epsilon > 0$).

One-way functions and permutations. One-way functions that are easy to compute but hard to invert on random inputs. The formal definition is as follows:

Definition 2.2.1 (One-way function). Let $f : \{0,1\}^* \to \{0,1\}^*$. We say that f is a *one-way function* if f is computable in polynomial-time, and for every polynomial-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$,

$$\Pr_{x\leftarrow_{\mathrm{R}}\{0,1\}^n}[C_n(f(x)) \in f^{-1}(f(x))] < \mathsf{neg}(n)$$

.

We say that f is a *one-way permutation* if f is a one-way-function, and for every $n \in \mathbb{N}$, f restricted to $\{0,1\}^n$ is a permutation of $\{0,1\}^n$.

Pseudorandom generators. A pseudorandom generator is is an algorithm that allows to take a small seed and expand it into a longer string that is computationally indistinguishable from the uniform distribution. The formal definition is as follows:

Definition 2.2.2. Let $l : \mathbb{N} \to \mathbb{N}$ be a polynomial-time computable function such that $l(n) > n$. A polynomial-time computable function $G : \{0,1\}^* \to \{0,1\}^*$ is a *pseudorandom generator* with expansion l, if for every string $s \in \{0,1\}^*$, $|G(s)| = l(|s|)$, and for every $n \in \mathbb{N}$, the random variables $G(U_n)$ and $U_{l(n)}$ are computationally indistinguishable.

The fundamental theorem about pseudorandom generators is the following:

Theorem 2.2.3 ([HILL89]). *Pseudorandom generators (with any polynomial expansion) exist if and only if one-way functions exist.*

Function ensembles and pseudorandom functions. We now define function ensembles and pseudorandom functions.

Definition 2.2.4 (Efficiently computable function ensembles.). An *function ensemble* is a family of functions $\{f_\alpha\}_{\alpha\in\{0,1\}^*}$ such that f_α is a function from $\{0,1\}^{|\alpha|}$ to $\{0,1\}^{|\alpha|}$.[1] We say that a function ensemble is *efficiently computable* if there exists a polynomial-time algorithm F such that given $\alpha, x \in \{0,1\}^n$, $F(\alpha, x) = f_\alpha(x)$.

The index α of a function ensemble is sometimes called the *seed* of the ensemble. We remark that we may sometimes identify the seed α with the function f_α. For example, we may say that a party samples a random function f from the ensemble and outputs $f(x)$, instead of saying that a party samples a random string α, and outputs $f_\alpha(x)$.

[1] One can also generalize the definition to allow α to belong to an arbitrary index set $I \subseteq \{0,1\}^*$ (where there is an efficient algorithm that given 1^n, samples a random element out of $I \cap \{0,1\}^n$), and we can also have f_α be a function from $\{0,1\}^{l_{\mathsf{in}}}(|\alpha|)$ to $\{0,1\}^{l_{\mathsf{out}}}(|\alpha|)$, where $l_{\mathsf{in}}(n), l_{\mathsf{in}}(n) = n^{O(1)}$.

Definition 2.2.5 (Pseudorandom functions). An efficiently computable function ensemble $\{f_\alpha\}_{\alpha\in\{0,1\}^*}$ is called a *pseudorandom function ensemble* if for every polynomial-sized oracle circuit family $\{C_n\}_{n\in\mathbb{N}}$, the following two random variables are computationally indistinguishable:

- $C^{f_\alpha}(1^n)$, where $\alpha \leftarrow_{\mathrm{R}} \{0,1\}^n$
- $C^H(1^n)$, where H is a function chosen at random from $\{0,1\}^n \to \{0,1\}^n$.

We remark that one can view pseudorandom functions as a pseudorandom generator with *exponential* expansion (with the seed to the generator being the choice of the member in the family). The basic theorem about pseudorandom functions is the following:

Theorem 2.2.6 ([GGM86]). *Pseudorandom function ensembles exist if and only if one-way functions exist.*

Trapdoor permutations. Loosely speaking, a trapdoor permutation ensemble is an efficiently computable permutation ensemble that is easy to compute, hard to invert, but easy to invert if you have additional trapdoor permutation. For simplicity, we define trapdoor permutations whose domain and range are equal to $\{0,1\}^n$. However, for all our applications it actually suffices to assume the existence of an *enhanced* family of trapdoor permutations [Gol01b, Appendix C.1]. Such a family can be constructed under either the RSA or factoring assumptions, see [ACGS84, Section 6.2] and [Gol01b, Appendix C.1].

Definition 2.2.7 (Trapdoor permutations). A *trapdoor permutation collection* is a set of three algorithms G, F, B (where G is probabilistic polynomial-time and F,B are deterministic polynomial-time) that satisfy:

- Let $(f,b) \leftarrow_{\mathrm{R}} G(1^n)$. Then, the function $p_f(x) \stackrel{def}{=} F(f,x)$ is a permutation on $\{0,1\}^n$ and the function $q_b(x) \stackrel{def}{=} B(b,x)$ is equal to p_f^{-1}.
- For every polynomial-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$,

$$\Pr_{(f,b)\leftarrow_{\mathrm{R}}\{0,1\}^n, x\leftarrow_{\mathrm{R}}\{0,1\}^n}[C_n(f,p_f(x)) = x] < \mathsf{neg}(n)$$

We say that the collection is *certified* if it possible to efficiently decide whether a string f was output by G. That is, if the language $L = \{f \mid (f,b) \in \mathrm{Supp}(G(1^*))\}$ is in **BPP**.

Again, also in the case of trapdoor permutations, we will sometimes identify the function computed with the seed value, and so say that a party samples two functions f, f^{-1} from a trapdoor permutation collection, instead of saying that a party computes $(f,b) \leftarrow_{\mathrm{R}} G(1^n)$.

Simple commitment schemes. A *commitment scheme* allows a party to digitally commit to a particular string/number, and then to reveal this value at a later time. It is a very useful building block for constructing secure protocols. There are many variants of commitment schemes, and some variants (such as interactive schemes, commit-with-extract, and non-malleable commitments) will be used also in this thesis. However, we'll start with defining the simplest notion, of a non-interactive perfectly binding commitment scheme.

Definition 2.2.8 (Simple commitment). A *(non-interactive perfectly-binding computationally-hiding) commitment schemes* is a polynomial-time computable sequence of functions $\{C_n\}_{n\in\mathbb{N}}$ where $C_n : \{0,1\}^n \times \{0,1\}^{p(n)} \to \{0,1\}^{q(n)}$, and $p(\cdot), q(\cdot)$ are some polynomials, that satisfies:

Perfect Binding For every $x \neq x' \in \{0,1\}^n$, $C(x, \{0,1\}^{q(n)}) \cap C(x', \{0,1\}^{q(n)}) = \emptyset$.

Computational Hiding For every $x, x' \in \{0,1\}^n$, the random variables $C(x, U_n)$ and $C(x', U_n)$ are computationally indistinguishable.

The first input x of the commitment scheme is sometimes called the *plaintext*. The second input is sometimes called the *random coins*.

We'll usually omit all the qualifiers, and just call such a scheme a *simple commitment scheme* (or sometimes, just a *commitment scheme*). A simple commitment scheme can be constructed under the assumption that one-way permutations exist [Blu82] (using the generic hard-core bit of [GL89]). Another construction, under incomparable assumptions, was given by [BOV03]. In all the places we use simple commitment schemes in this thesis, we can also use instead the two-round scheme of Naor [Nao89], which can be based on any one-way function.

We will sometimes denote a simple commitment scheme by Com. We will use the notation $\mathsf{Com}(x)$ to denote the random variable that is the result of committing to the plaintext x (and choosing the second input at random). We will also sometimes use the notion of a *bit commitment scheme*, where one commits to a single bit (and the security of the scheme is thus related not to the length of the plaintext but rather to an additional security parameter). Note that a string commitment scheme can be obtained from a bit commitment scheme by using the bit-commitment scheme to commit separately to each bit of the plaintext string.

2.2.1 Hashing and Tree-Hashing

We now define collision-resistant hash functions. It is reasonably conjectured that several efficient functions, such as SHA-1 [Nat95], give rise to such an ensemble. One can also construct such functions based on several natural hardness assumptions, such as the hardness of factoring [GMR84].

Definition 2.2.9. An efficiently computable function ensemble $\{h_\alpha\}_{\alpha\in\{0,1\}^*}$, where $h_\alpha : \{0,1\}^* \to \{0,1\}^{|\alpha|}$ is called *collision resistant* if for every polynomial-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$,

$$\Pr_{\alpha \leftarrow_{\mathrm{R}} \{0,1\}^n}[C_n(\alpha) = \langle x, y\rangle \text{ s.t. } x \neq y \text{ and } h_\alpha(x) = h_\alpha(y)] < \mathsf{neg}(n)$$

.

We defined hash functions as mapping strings of arbitrary length to n-bit long strings. One can also define hash functions as mapping n-bit strings to $n/2$-bit strings, or n-bit strings to n^ϵ-bit strings, where $1 > \epsilon > 0$ is some constant. It is not hard to see that a function ensemble satisfying one of these variants can be used to construct function ensembles satisfying the other variants. If $\{h_\alpha\}$ is a collision-resistent hash function ensemble, we will sometimes denote the set $\{h_\alpha \mid \alpha \in \{0,1\}^n\}$ by $\mathcal{H}_n$, and identify the hash function with its seed. For example, if we say that a party chooses a random $h \in \mathcal{H}_n$ and sends h, then we mean that this party chooses a random $\alpha \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends α.

Random-access hashing. Note that a collision-resistent hash function allows one party (which we'll call the *sender*) to *commit* to a string x by providing another party (which we'll call the *receiver*) with the value $y = h(x)$, where $h : \{0,1\}^* \to \{0,1\}^n$ is chosen by the receiver at random from the collection. This is a commitment in the sense that with overwhelming probability, if the sender sends a string y, then he can find at most one value x such that $y = h(x)$.[2] To decommit to a string y, a sender will send the string x such that $h(x) = y$. Suppose now that the receiver doesn't care about the entire string x, and only wants to know if x_i (i.e., the i^{th} bit of x) is equal to 0 or to 1. A *random-access hashing scheme* (also known as *tree hashing* or *Merkle's hash trees* [Mer89]) allows the sender to send a *certificate* to the value of x_i, in a way that the certificate size and time to verify it is polynomial in $|y|$ and $\log|x|$. This may be significantly shorter than simply sending the entire string x to the verifier. We now present the formal definition:

Definition 2.2.10 (Random-access hashing). A *random-access hashing collection* is an ensemble $\{\langle h_\alpha, \mathsf{cert}_\alpha\rangle\}_{\alpha\in\{0,1\}^*}$ of *pairs* of efficiently computable functions, where $h_\alpha : \{0,1\}^* \to \{0,1\}^{|\alpha|}$ and cert_α takes two inputs x, i, where $x \in \{0,1\}^*$ and $|i| = \log|x|$, and a polynomial-time algorithm V that satisfy the following properties:

Efficiency: $|\mathsf{cert}_\alpha(x,i)| = \mathrm{poly}(|\alpha|, \log|x|)$

Completeness: For every α, x, $V_{\alpha,h_\alpha(x)}(i, x_i, \mathsf{cert}_\alpha(x,i)) = 1$.[3]

Binding (Soundness): For every polynomial-size circuit family $\{C_n\}_{n\in\mathbb{N}}$,

$$\Pr_{\alpha\leftarrow\{0,1\}^n}[C_n(\alpha) = \langle y, i, \sigma_0, \sigma_1\rangle \text{ s.t. } V_{\alpha,y}(i,0,\sigma_0) = 1 \text{ and } V_{\alpha,y}(i,1,\sigma_1) = 1]$$

Note that if we ignore the second function then a random-access hashing ensemble is in particular a collision-resistent hash function ensemble. Indeed, suppose that there is an algorithm that given α obtains a collision with respect to h_α (i.e., $x \neq x'$ such that $h_\alpha(x) = h_\alpha(x')$). Then, we can convert this algorithm to an algorithm contradicting the binding property of the random-access hashing scheme. Indeed, there exists i such that (without loss of generality) $x_i = 0$ and $x'_i = 1$. This means that if we let $y = h_\alpha(x) = h_\alpha(x')$, $\sigma^0 = \mathsf{cert}_\alpha(x,i)$, and $\sigma^1 = \mathsf{cert}_\alpha(x',i)$ then $V_{\alpha,y}(i,0,\sigma^0) = 1$ and $V_{\alpha,y}(i,1,\sigma^1) = 1$.

Constructing a random-access hashing scheme using hash trees. There is a well known construction due to Merkle of a random-access hash scheme based on any collision-resistent hash function ensemble [Mer89]. Let $\{h'_\alpha\}_{\alpha\in\{0,1\}^*}$ be a collision-resistent hash function ensemble, where where $h'_\alpha : \{0,1\}^* \to \{0,1\}^{|\alpha|}$ (actually it is enough to use such a collection where $h'_\alpha : \{0,1\}^{2|\alpha|} \to \{0,1\}^{|\alpha|}$). The random-access hashing scheme $\{h_\alpha, \mathsf{cert}_\alpha\}_{\alpha\in\{0,1\}^*}$ will be defined as follows:

The hash function h_α: Given an input x, one pads x in some canonical way to a string of length $2^d n$. That is, we assume $x = x_1 \cdots x_{2^d}$ where $x_i \in \{0,1\}^n$ (and $n = |\alpha|$). To compute $h_\alpha(x)$, we construct a depth-d complete binary tree, which we label iteratively, from the leaves to the root. Each one of the 2^d leaves is labeled with the corresponding block x_i. We then iteratively label each node in the tree with the hash of its children's labels. That is, if we denote the label of a node v by L_v, and we denote the left (resp. right) child of a node v by $\mathsf{left}(v)$ (resp. $\mathsf{right}(v)$), then $L_v = h'_\alpha(L_{\mathsf{left}(v)} \circ L_{\mathsf{right}(v)})$ (where $\circ$ denotes concatenation). We let $h_\alpha(x)$ to be $\langle |x|, L\rangle$ where L is the label of the root of the binary tree.

[2]We say that a hash function is like a commitment only in the sense that it satisfies a computational binding property. We do not make any secrecy/hiding requirements from the hash function.

[3]Note that we use subscript notation for the first two inputs to V.

The certification function $\mathsf{cert}_\alpha(i,x)$: To compute $\mathsf{cert}_\alpha(i,x)$ we compute the same labeled binary tree as in the computation of $h_\alpha(x)$. Let i' denote the index of the block of the padded x that contains the i^{th} bit of x. We define $\mathsf{cert}_\alpha(x)$ to be the set of all labels of the nodes along the path from the leaf corresponding to the block $x_{i'}$ to the root, including the labels for the immediate siblings of these nodes.

The verification algorithm V: Given a seed α, a value y (which supposed to be the label of the root of the tree), an index i, a bit $b \in \{0,1\}$, and a certificate σ (which contains a list of labels), the verification algorithm is as follows: Initially V computes the index i' of the block that contains the i^{th} bit of x. Then V checks that the bit corresponding to the i^{th} bit of x in the block $x_{i'}$ (which is part of the certificate σ) is indeed equal to b (otherwise it outputs 0). Then, V verifies that for every non-leaf label L_v in the path between $x_{i'}$ to the root, it holds that $L_v = h'_\alpha(L_{\mathsf{left}(v)} \circ L_{\mathsf{right}(v)})$ (note that the labels $L_{\mathsf{left}(v)}$ and $L_{\mathsf{right}(v)}$ should indeed be part of the certificate σ); otherwise V outputs 0. Finally, V verifies that the label for the root in σ is indeed equal to y.

The completeness and efficiency conditions of this scheme are pretty self-evident. The reason this scheme satisfies the binding condition is the following: if one compares a valid certificate for $x_i = 0$ and a valid certificate for $x_i = 1$ (with respect to the same seed α and root label y), then both these certificates contain labels for all the nodes in the path from the root to the block $x_{i'}$. Because the label for the root is equal in both certificates, and the label for the leaf $x_{i'}$ is different, there must be some node v in the path such that $L_v = L'_v$ but either $L_{\mathsf{left}(v)} \neq L'_{\mathsf{left}(v)}$ or $L_{\mathsf{right}(v)} \neq L'_{\mathsf{right}(v)}$ (where L denotes the label according to the first certificate and L' denotes the label according to the second certificate). We see that if we denote $u = L_{\mathsf{left}(v)} \circ L_{\mathsf{right}(v)}$ and $u' = L'_{\mathsf{left}(v)} \circ L'_{\mathsf{right}(v)}$, then $u \neq u'$ but $h'_\alpha(u) = h'_\alpha(u')$: that is, one can derive a collision for h'_α from these two certificates.

2.3 Interactive Protocols, Interactive Proofs and Zero Knowledge

2.3.1 Protocols and interaction

In many cases, we will be interested in this thesis in an *interactive* setting, where there are two or more parties that are exchanging messages between them.[4] We assume that the reader is familiar with the notions of interactive algorithms and protocols, and thus our emphasis is on presenting the notations that we will use. See [Gol01b] for more elaborate and precise descriptions of interactive protocols and interactive Turing machines.

Next-message function, view and transcript. A *view* of a party in an interaction contains the public input, the party's private input and random-tape, and the list of messages that this party received up to this step. The *next-message function* of a party is a function that maps a view in a particular step of the protocol to the party's message in the next step. An *interactive algorithm* is an algorithm that computes the next-message function of a party. If I is an interactive algorithm, and v is a view of the protocol up to a particular step s, then the *residual algorithm I with respect to the view v* is the algorithm I with the view v "hardwired in". That is, this is a function that takes a list of messages sent after the step s, and computes I's response to them, assuming that the messages sent up to step s are as described in v. A *transcript* of an interaction

[4] Actually, we will almost always be interested in the case where there are exactly two parties.

consists of the public input and the list of all messages exchanged in the interaction (but does not include the parties' private inputs and random-tapes). That is, the transcript of an execution is the public information seen by both parties in the execution. For any interactive algorithm I, and any view v of I, the transcript τ of the protocol can always be computed from the view v. We say in this case that the transcript τ is *contained* in the view v.

Notation. If A and B are interactive algorithms, then $\langle A(y), B(z)\rangle(x)$ is a random variable representing the execution of an interaction between A and B on public input x when A's private input is y and B's private input is z. We let $\mathsf{view}_A\langle A(y), B(z)\rangle(x)$ denotes A's view in this execution, and let $\mathsf{out}_A\langle A(y), B(z)\rangle(x)$ denotes A's output at the end of the execution. We define view_B and out_B in the symmetric way. We let $\mathsf{transcript}\langle A(y), B(z)\rangle(x)$ denote the transcript of the execution.

Prescribed versus cheating parties. A *protocol* is a set of interactive algorithms, which determines the strategies that each player is supposed to use. We call these strategies the *prescribed* or *honest* strategies. However, we will usually analyze the execution of the protocol also when one of the parties may *not* be following its prescribed strategy. We will sometimes refer to such a party as "cheating". Part of the description of any protocol are conventions on who is the first party to send a message and on the length of each message. We assume, without loss of generality, that even "cheating" parties follow these conventions (e.g., if a party sends a message that is too short or too long then we assume that it is padded or truncated to the proper length).

Hardwiring input convention. We will usually assume that cheating parties do *not* get any private input. This will be without loss of generality because our adversarial model is non-uniform circuits. Thus, any input that a cheating party may have can be considered as being "hardwired" into it, and so part of its description as a Boolean circuit.

2.3.2 Interactive proof and argument systems

An *interactive proof* [GMR85] is a two-party protocol, where one party is called the *prover* and the other party is called the *verifier*. We use the following definition:

Definition 2.3.1 (Interactive proofs). An interactive protocol (P, V) is called an *interactive proof system* for a language L if the following conditions hold.

Efficiency: The number and total length of messages exchanged between P and V are polynomially bounded and V is a probabilistic polynomial-time machine.

Perfect completeness: If $x \in L$, then $\Pr[\mathsf{out}_V\langle P, V\rangle(x) = 1] = 1$.

Soundness: If $x \notin L$, then for every possible P^*, $\Pr[\mathsf{out}_V\langle P^*, V\rangle(x) = 1] \leq \mathsf{neg}(n)$.

An interactive proof system is called *Arthur-Merlin* [BM88] (a.k.a. *public-coins*) if the verifier's messages consist only of random strings and acceptance is computed as a deterministic polynomial-time function of the interaction's transcript. An interactive proof system that is not Arthur-Merlin is called *private-coins*.

The number of *rounds* in an interactive proof is the total number of messages exchanged in the interaction (that is, both prover messages and verifier messages).

Let $L \in \mathbf{NP}$, a proof system for L has an *efficient prover strategy* if the completeness property of the system can be satisfied by a probabilistic polynomial-time algorithm that when proving that $x \in L$ gets as auxiliary input a witness to this fact.

Let $L \in \mathbf{NP}$, an *interactive argument* for L [BCC88] is the following variation on the definition of an interactive proof:

- The soundness requirement is relaxed to quantify only over prover strategies P^* that can be implemented by a polynomial-sized circuit.
- The system is required to have an efficient prover strategy.

Honest verifier conventions. We say that an execution of a two-party protocol is *completed successfully* if no party aborted. For a proof (or argument) system, we assume that if a verifier rejects the proof then it aborts, and so an execution of a proof system is completed successfully only if the verifier accepts. If a proof system uses another proof system as a subprotocol, then we assume that in case the verifier of the subprotocol rejects, then execution is aborted and thus the verifier for the larger proof system will also reject. If we do not specify the verifier's acceptance condition, then it is assumed that the verifier accepts if and only if all the subprotocols were completed successfully.

2.3.3 Zero-knowledge

Informally, we say that a proof/argument system for L is *zero-knowledge* [GMR85] if after seeing a proof that $x \in L$, the verifier does not learn anything about x that it didn't know before. We require this to hold even if the verifier does not follow its prescribed strategy for the proof system, as long as its strategy can be implemented by an efficient algorithm. This is formalized by requiring that there exists an efficient algorithm called the *simulator*, that given the verifier's prior knowledge (i.e., the string x, the verifier's strategy and private inputs) can compute (or closely approximate) the verifier's state after viewing a proof that $x \in L$. The formal definition is below. Note that we define two variants of zero-knowledge: *uniform* and *non-uniform*, based on the classes of algorithms that we allow the cheating verifier to employ. Note also that in our definition we require the simulator to be *universal*. That is, the simulator is a single algorithm for all possible verifier's strategies, that gets the strategy as an additional input.

Definition 2.3.2 (Zero-knowledge). Let $L = L(R)$ be some language and let (P, V) be an interactive argument for L. We say that (P, V) is *(non-uniform) zero-knowledge* if there exists a probabilistic polynomial-time algorithm S such that for every polynomial-sized circuit family $\{V_n^*\}_{n\in\mathbb{N}}$ and every sequence $\{(x_n, y_n)\}_{n\in\mathbb{N}}$, where $x_n \in \{0,1\}^n \cap L$ and $(x_n, y_n) \in R$, the following two probability ensembles are computationally indistinguishable:

- $\left\{ \mathsf{view}_{V_n^*}\langle P(y_n), V_n^*\rangle(x_n) \right\}_{n\in\mathbb{N}}$
  and
- $\left\{ S(V_n^*, x_n) \right\}_{n\in\mathbb{N}}$

We say that (P, V) is *uniform zero-knowledge* if there exists a probabilistic polynomial-time algorithm S such that for every polynomial $t(\cdot)$, every probabilistic $t(n)$-time Turing machine V^* and every sequence $\{(x_n, y_n)\}_{n\in\mathbb{N}}$, where $x_n \in \{0,1\}^n \cap L$ and $(x_n, y_n) \in R$ the following two probability ensembles are computationally indistinguishable:

- $\left\{ \mathsf{view}_{V^*}\langle P(y_n), V^*\rangle(x_n) \right\}_{n\in\mathbb{N}}$
  and

- $\left\{S(V^*, 1^{t(n)}, x_n)\right\}_{n\in\mathbb{N}}$

In either case we say that S is a *black-box* simulator if the only use it makes of its first input (i.e., V^*) is to call it as a subroutine.

We remark that we use a somewhat stronger definition than the standard definition of uniform zero-knowledge [Gol93]: we allow both the input generation and the distinguisher to be non-uniform.

2.3.4 Witness indistinguishability

Like zero-knowledge, a *witness-indistinguishable* proof/argument system [FS90] also guarantees some secrecy property to the prover, but it is a weaker property than zero-knowledge. In a witness-indistinguishable proof system we do not require that the verifier does not learn anything about x after seeing a proof that $x \in L$. Rather, we only require that if both y and y' are witnesses that $x \in L$, then it is infeasible for the verifier to distinguish whether the prover used y or y' as auxiliary input. The formal definition is below: (we only make the definition in the non-uniform setting)

Definition 2.3.3. Let $L = L(R)$ be some language and let (P, V) be an interactive argument for L. We say that (P, V) is *witness-indistinguishable* if for every polynomial-sized circuit family $\{V_n^*\}_{n\in\mathbb{N}}$ and every sequence $\{(x_n, y_n, y_n')\}_{n\in\mathbb{N}}$, where $x_n \in \{0,1\}^n$ and $(x_n, y_n), (x_n, y_n') \in R$ the following two probability ensembles are computationally indistinguishable:

- $\left\{\mathsf{view}_{V_n^*}\langle P(y_n), V_n^*\rangle(x_n)\right\}_{n\in\mathbb{N}}$

  and
- $\left\{\mathsf{view}_{V_n^*}\langle P(y_n'), V_n^*\rangle(x_n)\right\}_{n\in\mathbb{N}}$

Witness indistinguishability is a weaker property than zero-knowledge. That is, if a protocol is zero-knowledge then it is also witness-indistinguishable [FS90]. Also, under standard assumptions, there exist protocols that are witness-indistinguishable but *not* zero-knowledge (as a trivial example, note that for any language L where each $x \in L$ has a single witness, the trivial **NP** proof system of sending the witness is witness-indistinguishable). Unlike zero-knowledge, witness indistinguishability is known to be closed under concurrent (and in particular parallel) composition [Fei90]. Using this fact, parallel repetition of the "basic protocol" of [GMW86], yields the following theorem:

Theorem 2.3.4 ([FS90]). *Suppose that one-way functions exist. Then, for every language $L \in$* **NP** *there exist a constant-round Arthur-Merlin witness-indistinguishable proof system for L.*

Proofs of knowledge

In a proof/argument system, the prover convinces the verifier that some string x is a member of a language L. In a proof/argument *of knowledge* [FFS87, BG93, GMR85, TW87] the prover should convince the verifier that it also *knows* a witness to the fact that $x \in L$. This is formalized by requiring that if the verifier is convinced with some probability p by some (possibly cheating) prover strategy, then by applying an efficient algorithm, called the *knowledge extractor*, to the cheating prover's strategy and private inputs, it is possible to obtain a witness to the fact that $x \in L$, with probability (almost equal to) p. The formal definition is below:

Definition 2.3.5. Let $L = L(R)$ and let (P, V) be an argument system for L. We say that (P, V) is an *argument of knowledge* for L if there exists a probabilistic polynomial-time algorithm E (called the *knowledge extractor*) such that for every polynomial-sized prover strategy P^* and for every $x \in \{0,1\}^n$, if we let $p_* \stackrel{def}{=} \Pr[\mathsf{out}_V \langle P^*, V\rangle(x) = 1]$ then

$$\Pr[E(P^*, x) \in R(x)] \geq p_* - \mathsf{neg}(n)$$

.

We say that an argument of knowledge has a *black-box* extractor if the knowledge extractor algorithm E uses its first input (i.e., P^*) as a black-box subroutine (i.e., oracle).

We will sometimes consider a generalized definition where we allow both the cheating prover P^* and the extractor E to run in time $T(n)^{O(1)}$ where $T(\cdot)$ is some *super-polynomial* function.

2.4 Universal Arguments

Universal arguments are a variant of (interactive) CS proofs, as defined and constructed by Micali [Mic94] and Kilian [Kil92]. Loosely speaking, a *universal argument* is an interactive argument of knowledge for proving membership in **NEXP**. Note that all of the languages in **NP** can be reduced to a language in **NEXP** via a reduction that preserves the length of the instance. Therefore, an argument system for **NEXP** allows us to use a single protocol to prove membership in *all* **NP** languages, rather than use a different protocol for each language; hence the name *universal arguments.*

For sake of simplicity, we define and present universal argument systems only for the following *universal language* $L_{\mathcal{U}}$:[5] the tuple $\langle M, x, t\rangle$ is in $L_{\mathcal{U}}$ if M is a non-deterministic machine that accepts x within t steps. Clearly, every **NP**-language L is linear-time reducible to $L_{\mathcal{U}}$ (i.e., via the reduction $x \mapsto \langle M_L, x, 2^{|x|}\rangle$, where M_L is any fixed non-deterministic polynomial-time deciding L). Thus, a proof system for $L_{\mathcal{U}}$ allows us to handle all "NP-statements" (in a uniform manner); that is, there exists a single polynomial p such that for every $L \in$ **NP**, the complexity of verifying that $x \in L$ is bounded by $p(|x|)$. In fact, $L_{\mathcal{U}}$ is **NE**-complete (by an analogous linear-time reduction).[6] We consider also the natural witness-relation for $L_{\mathcal{U}}$, denoted $R_{\mathcal{U}}$: the pair $(\langle M, x, t\rangle, w)$ is in $R_{\mathcal{U}}$ if M (viewed here as a two-input deterministic machine) accepts (x, w) within t steps.

We also require a universal argument system to satisfy a *proof of knowledge* property. However, because we wish to have a polynomial-time knowledge extractor, but the witness size may be exponential, the extractor may not be able to write down the witness. Thus, we only require the extractor to output an *implicit representation* for the witness (i.e., a circuit that on input i outputs the i^{th} bit of the witness). We also relax the proof of knowledge property and not require the knowledge extractor to output a witness with probability equal (or very close) to p^*, where p^* is the probability that the verifier is convinced to accept the proof. Rather, it is sufficient for our purposes that the extractor outputs a witness with probability that is polynomially related to p^*.

Definition 2.4.1 (Universal arguments). A *universal-argument system* is a pair of strategies, denoted (P, V), that satisfies the following properties:

[5] The nice aspect about $L_{\mathcal{U}}$ is that it comes with a natural measure of complexity of instances: the complexity of (M, x, t) is the actual time it takes M to accept x (when using either the best or a given sequence of non-deterministic choices). Such a complexity measure is pivotal to the refined formulation of the prover complexity condition.

[6] Furthermore, every language in **NEXP** is polynomial-time (but not linear-time) reducible to $L_{\mathcal{U}}$.

Efficient verification: There exists a polynomial p such that for any $y = \langle M, x, t\rangle$, the total time spent by the *(probabilistic)* verifier strategy V, on common input y, is at most $p(|y|)$. In particular, all messages exchanged in the protocol have length smaller than $p(|y|)$.

Completeness by a relatively-efficient prover: For every $(\langle M, x, t\rangle, w)$ in $R_{\mathcal{U}}$,

$$\Pr[\langle P(w), V\rangle(M, x, t) = 1] \ = \ 1$$

Furthermore, there exists a polynomial p such that the total time spent by $P(w)$, on common input $\langle M, x, t\rangle$, is at most $p(T_M(x, w)) \leq p(t)$.

Computational Soundness: For every polynomial-size circuit family $\{\widetilde{P}_n\}_{n\in\mathbb{N}}$, and every $\langle M, x, t\rangle \in \{0,1\}^n \setminus L_{\mathcal{U}}$,

$$\Pr[\langle \widetilde{P}_n, V\rangle(M, x, t) = 1] \ < \ \mathsf{neg}(n)$$

A weak Proof of Knowledge Property: For every positive polynomial p there exists a positive polynomial p' and a probabilistic polynomial-time oracle machine E such that the following holds:[7]

For every polynomial-size circuit family $\{\widetilde{P}_n\}_{n\in\mathbb{N}}$, and every sufficiently long $y = \langle M, x, t\rangle \in \{0,1\}^*$ if $\Pr[(\widetilde{P}_n, V)(y) = 1] > 1/p(|y|)$ then then

$$\Pr[E^{\widetilde{P}_n}(y) = C \text{ s.t. } [C] \in R_{\mathcal{U}}(y)] > \frac{1}{p'(|y|)}$$

(where $[C]$ denotes the function computed by the Boolean circuit C).

The oracle machine E is called a *(knowledge) extractor*.

There are two differences between universal arguments and (interactive) CS-Proofs [Mic94]:

1. The computational soundness in CS-Proofs needs to hold for cheating provers of size $poly(t)$. In contrast, the computational soundness in universal arguments needs only to hold for adversaries of size polynomial in the input length which is $|M| + |x| + \log(|t|)$.
2. We require that a universal argument satisfies a *proof of knowledge* condition. Note that we allow the knowledge extractor only polynomial time in the input length which is $|M| + |x| + \log(|t|)$, while the size of the witness may be t which is may be exponentially larger than the input length. Therefore, the knowledge extractor can output only an implicit representation of the witness.

We will use the following (almost trivial) lemma:

Lemma 2.4.2. *Fix $f : \mathbb{N} \to \mathbb{N}$ to be some super-polynomial function (e.g., $f(n) = n^{\log n}$). Then, there exists an extractor algorithm E' that on input $\langle M, x, t\rangle$ and oracle access to a polynomial-sized circuit $\tilde{P}$ that such that $\Pr[\langle \widetilde{P}_n, V\rangle(M, x, t) = 1] > \frac{1}{f(n)}$ runs in time $(f(n) \cdot t)^{O(1)}$ and outputs a witness $w \in R_{\mathcal{U}}(M, x, t)$ with probability $1 - \mathsf{neg}(n)$.*

Proof Sketch: Using $t \cdot poly(n)$ steps, it is possible to convert an implicit representation of a witness into an explicit one. Thus by invoking the extractor E from the proof of knowledge property $n \cdot f(n)$ times and doing this conversion each time we can achieve the desired result. □

[7] Indeed, the polynomial p' as well as the (polynomial) running-time of E may depend on the polynomial p (which defines the noticeable threshold probability above).

Note: Lemma 2.4.2 implies that if we use the universal arguments system to prove statements in $\mathbf{Ntime}(T(n))$, where $T(n)$ is some super-polynomial function, and we allow both the cheating prover and the knowledge extractor to run in time $T(n)^{O(1)}$, then we get a stronger proof of knowledge condition. That is, the knowledge extractor obtains a witness (and not just an implicit representation of it) and can amplify the probability of extraction. In many places in this thesis, where we assume that there our primitives are secure against $T(n)$-sized circuits, we will make use of this implication.

The following theorem implies that universal arguments exist under standard assumptions:

Theorem 2.4.3 ([BG01]). *Suppose that collision-resistent hash functions exist. Then there exists a universal argument system.*

And the following stronger variant.

Theorem 2.4.4 ([BG01]). *Suppose that collision-resistent hash functions exist. Then for every* $\epsilon > 0$*, there exists a constant-round universal argument system in which the total size of the messages exchanged when proving a statement of length* n *verifiable in time* t *is at most* $n^\epsilon \mathrm{polylog}(t)$*. Furthermore,*

1. *There exist such systems where the prover is zero-knowledge.*
2. *There exist such systems that are Arthur-Merlin (public-coins) and the prover is witness indistinguishable (WI).*

We prove Theorems 2.4.3 and 2.4.4 in Appendix A. We stress that these theorems only require *standard* collision-resistent hash functions (i.e., secure against polynomial-sized circuits). However, it is not hard to see that if we assume that there exist hash functions that are collision resistent against $T(n)^{O(1)}$-sized circuits, then there exists such systems in which the soundness, the proof of knowledge and zero-knowledge property hold against $T(n)$-sized circuits.

2.5 Commit-with-Extract Schemes

Commit-with-extract schemes. Loosely speaking, a *commit-with-extract* scheme [BL02] is a commitment scheme where the sender also proves the knowledge of the committed value. The formal definition of commit-with-extract follows the definition of "witness-extended emulation"[8] [Lin01] and requires that there exists an extractor that can simulate the view of the sender, and output a committed value that is compatible with this view. (See [BL02] for more discussions about the definition of commit-with-extract.) The resulting definition is the following:

Definition 2.5.1 (Interactive commitment scheme). A two-party protocol (S, R) and an algorithm R' (where S is called the *sender* and R is called the *receiver*) is a *(statistically-binding) computationally hiding) interactive commitment scheme* if it satisfies the following properties:

Completeness with public decommitment Let τ be the transcript of an execution between S and R, where S gets $x \in \{0,1\}^n$ as a private input, R has no private input, and the public input is 1^n, and let r_S be the randomness used by S in this execution. Then, $R'(\tau, x, r_S) = 1$ (which means that the receiver accepts r_S as a valid decommitment for x).[9]

[8] A *witness extended emulator* is a knowledge extractor that outputs a simulated transcript along with the extracted witness.

[9] Note that R' does not need to see the randomness used by R in the execution, but only needs the public transcript τ. This is why we call this condition "completeness with public decommitment". All the commitment schemes in this paper satisfy this condition.

Computational hiding For every $x, x' \in \{0,1\}^n$, the random variables $\mathsf{view}_R\langle S(x), R\rangle(1^n)$ and $\mathsf{view}_R\langle S(x'), R\rangle(1^n)$ are computationally indistinguishable.

Statistical binding For every (possibly cheating) polynomial-sized S^*, if τ is the random variable denoting an execution between S^* and R on public input 1^n, then the probability that there exist $x \neq x' \in \{0,1\}^n$ and $r, r' \in \{0,1\}^*$ such that $R'(\tau, x, r) = 1$ and $R'(\tau, x', r') = 1$ is at most $\mathsf{neg}(n)$.[10]

We will usually drop the qualifiers and simply call a scheme satisfying Definition 2.5.1 a *commitment scheme*. Note that Definition 2.5.1 is a generalization of Definition 2.2.8 in the sense that they are equivalent for non-interactive protocols (i.e., for the case where R doesn't do anything and S only sends one message).

The commitment value function. Let us fix a commitment scheme (S, R), R'. We define the following (non-efficiently computable) function $\mathsf{com\text{-}value}$: on input a transcript τ, $\mathsf{com\text{-}value}(\tau)$ returns the unique value x for which there exists r such that $R'(\tau, x, r) = 1$ (if there is such a unique x). If there exists no such value x or if there exists more than one such value x then we define $\mathsf{com\text{-}value}(\tau)$ to be equal to $\perp$.

Definition 2.5.2 (commit with extract). A commitment scheme (S, R) , R' is a *commit-with-extract* scheme if there exists a probabilistic polynomial-time two-output algorithm CK (called the *commitment extractor*) such that for every polynomial-sized (possibly cheating) sender S^*, if we denote $(V, X) = CK(S^*, 1^n)$ then

Emulation The random variable V is computationally indistinguishable from the view of S^* in an interaction with the honest receiver R on public input 1^n.

Extraction With probability at least $1 - \mathsf{neg}(n)$, either $\mathsf{com\text{-}value}(T) = X$ or $\mathsf{com\text{-}value}(T) = \perp$, where T is the transcript contained in the view V.

A commit-with-extract scheme is called *liberal* if the extractor CK runs in *expected* probabilistic-polynomial-time. We say that CK is *black-box* if the only use it makes of its first input S^* is to call it as a black-box subroutine.

A constant-round liberal commit-with-extract scheme can be obtained based on any one-way-function by sequentially executing first a standard commitment scheme and then a constant-round zero-knowledge proof-of-knowledge for **NP** (e.g., the system of [FS89]) to prove knowledge of the commitment value. The commitment extractor CK will use the knowledge extractor of the commitment scheme. In this way, one obtains a *black-box* commitment extractor that rewinds the sender A in order to obtain the committed value. In Chapter 5, we will present a different construction of a commit-with-extract, that has a *strict* probabilistic polynomial-time *non-black-box* extractor.

[10] We call this property *statistical* binding even though it only holds with respect to efficient senders S^* because, after the interaction is finished, even an all powerful sender will only be able to cheat with negligible probability.

Chapter 3

Software Obfuscation

Summary: Informally, an *obfuscator* $\mathcal{O}$ is an (efficient, probabilistic) "compiler" that takes as input a program (or circuit) P and produces a new program $\mathcal{O}(P)$ that has the same functionality as P yet is "unintelligible" in some sense. Obfuscators, if they exist, would have a wide variety of cryptographic and complexity-theoretic applications, ranging from software protection to homomorphic encryption to complexity-theoretic analogues of Rice's theorem. Most of these applications are based on an interpretation of the "unintelligibility" condition in obfuscation as meaning that $\mathcal{O}(P)$ is a "virtual black box," in the sense that anything one can efficiently compute given $\mathcal{O}(P)$, one could also efficiently compute given oracle access to P.

In this work, we initiate a theoretical investigation of obfuscation. Our main result is that, even under very weak formalizations of the above intuition, obfuscation is impossible. We prove this by constructing a family of functions $\mathcal{F}$ that are *unobfuscatable* in the following sense: there is a property $\pi : \mathcal{F} \to \{0,1\}$ such that (a) given *any program* that computes a function $f \in \mathcal{F}$, the value $\pi(f)$ can be efficiently computed, yet (b) given *oracle access* to a (randomly selected) function $f \in \mathcal{F}$, no efficient algorithm can compute $\pi(f)$ much better than random guessing.

We extend our impossibility result in a number of ways, including even obfuscators that (a) are not necessarily computable in polynomial time, (b) only *approximately* preserve the functionality, and (c) only need to work for very restricted models of computation ($\mathbf{TC_0}$). We also rule out several potential applications of obfuscators, by constructing "unobfuscatable" signature schemes, encryption schemes, and pseudorandom function families.

3.1 Introduction

The past two decades of cryptography research has had amazing success in putting most of the classical cryptographic problems — encryption, authentication, protocols — on complexity-theoretic foundations. However, there still remain several important problems in cryptography about which theory has had little or nothing to say. One such problem is that of *program obfuscation.* Roughly speaking, the goal of (program) obfuscation is to make a program "unintelligible" while preserving its functionality. Ideally, an obfuscated program should be a "virtual black box," in the sense that anything one can compute from it one could also compute from the input-output behavior of the program.

The hope that some form of obfuscation is possible arises from the fact that analyzing programs expressed in rich enough formalisms is hard. Indeed, any programmer knows that total unintel-

This chapter is based on the paper [BGI$^+$01] which is joint work with Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan and Ke Yang. Note that the paper [BGI$^+$01] contains some additional results which are not included in this thesis.

ligibility is the natural state of computer programs (and one must work hard in order to keep a program from deteriorating into this state). Theoretically, results such as Rice's Theorem and the hardness of the Halting Problem and Satisfiability all seem to imply that the only useful thing that one can do with a program or circuit is to run it (on inputs of ones choice). However, this informal statement is, of course, highly speculative, and the existence of obfuscators requires its own investigation.

To be a bit more clear (though still informal), an *obfuscator* $\mathcal{O}$ is an (efficient, probabilistic) "compiler" that takes as input a program (or Boolean circuit) P and produces a new program $\mathcal{O}(P)$ satisfying the following three conditions:

Functionality: $\mathcal{O}(P)$ computes the same function as P.

Efficiency: $\mathcal{O}(P)$ is at most polynomially slower than P.

"Virtual black box" property: "Anything that can be efficiently computed from $\mathcal{O}(P)$ can be efficiently computed given oracle access to P."

While there are heuristic approaches to obfuscation in practice (cf., Figure 3.1 and [CT00]), there has been little theoretical work on this problem. This is unfortunate, since obfuscation, if it were possible, would have a wide variety of cryptographic and complexity-theoretic applications.

```
#include<stdio.h> #include<string.h>
main(){char*O,l[999]="'`acgo\177~|xp .
-\0R^8)NJ6%K4O+A2M(*0ID57$3G1FBL";
while(O=fgets(l+45,954,stdin)){*l=O[
strlen(O)[O-1]=0,strspn(O,l+11)];
while(*O)switch((*l&&isalnum(*O))-!*l)
{case-1:{char*I=(O+=strspn(O,l+12)
+1)-2,O=34;while(*I&3&&(O=(O-16<<1)+
*I---'-')<80);putchar(O&93?*I
&8||!(  I=memchr( l , O , 44 ) ) ?'?':
I-l+47:32); break; case 1: ;}*l=
(*O&31)[l-15+(*O>61)*32];while(putchar
(45+*l%2),(*l=*l+32>>1)>35); case 0:
putchar((++O ,32));}putchar(10);}}
```

Figure 3.1: The winning entry of the 1998 *International Obfuscated C Code Contest*, an ASCII/Morse code translator by Frans van Dorsselaer [vD98] (adapted for this paper).

In this chapter, we initiate a theoretical investigation of obfuscation. We examine various formalizations of the notion, in an attempt to understand what we can and cannot hope to achieve. Our main result is a negative one, showing that obfuscation (as it is typically understood) is *impossible*. Before describing this result and others in more detail, we outline some of the potential applications of obfuscators, both for motivation and to clarify the notion.

3.1.1 Some Applications of Obfuscators

Software Protection. The most direct applications of obfuscators are for various forms of software protection. By definition, obfuscating a program protects it against reverse engineering. For example, if one party, Alice, discovers a more efficient algorithm for factoring integers, she may wish to sell another party, Bob, a program for apparently weaker tasks (such as breaking the RSA cryptosystem) that use the factoring algorithm as a subroutine without actually giving Bob a factoring algorithm. Alice could hope to achieve this by obfuscating the program she gives to Bob.

Indeed, protecting software against reverse engineering has been the main application in mind for the commercially produced obfuscators (c.f., [Clo03]). Many of these applications are related to the area of Digital Rights Management (DRM). For example, obfuscation has been used (unsuccessfully) for the CSS (Content Scrambling System) for copy-protecting DVDs.[1] Also, recently the United States Air Force Research Laboratories awarded a $1.8M contract for research in obfuscation (among other software protection methods) [Net03].

Intuitively, obfuscators would also be useful in *watermarking* software (cf., [CT00, NSS99]). A software vendor could modify a program's behavior in a way that uniquely identifies the person to whom it is sold, and then obfuscate the program to guarantee that this "watermark" is difficult to remove.

Homomorphic Encryption. A long-standing open problem in cryptography is whether *homomorphic* encryption schemes exist (cf., [RAD78, FM91, DDN91, BL96, SYY99]). That is, we seek a secure public-key cryptosystem for which, given encryptions of two bits (and the public key), one can compute an encryption of any binary Boolean operation of those bits. Obfuscators would allow one to convert any public-key cryptosystem into a homomorphic one: use the secret key to construct an algorithm that performs the required computations (by decrypting, applying the Boolean operation, and encrypting the result), and publish an obfuscation of this algorithm together with the public key.[2]

Removing Random Oracles. The *Random Oracle Model* [BR93] is an idealized cryptographic setting in which all parties have access to a truly random function. It is (heuristically) hoped that protocols designed in this model will remain secure when implemented using an efficient, publicly computable cryptographic hash function in place of the random function. While it is known that this is not true in general [CGH98], it is unknown whether there exist efficiently computable functions with strong enough properties to be securely used in place of the random function in various *specific* protocols (e.g., in Fiat-Shamir type schemes for specific natural protocols [FS86]). One might hope to obtain such functions by obfuscating a family of pseudorandom functions [GGM86], whose input-output behavior is by definition indistinguishable from that of a truly random function.

[1]See, e.g., David Wagner's declaration on DVD CCA v. McLaughlin, Bunner, et al. case [Wag00]. A particularly relevant quote is the following: "My understanding is that the DVD security design relies in part on distributing software in an "obscured" form - hidden in locations that are not obvious. This cannot and does not prevent reverse engineering; it can make the reverse engineering task more tedious, but it is widely known that such obfuscation can be overcome by patience, talent, or sufficiently sophisticated reverse engineering tools."

[2]There is a subtlety here, caused by the fact that encryption algorithms must be *probabilistic* to be semantically secure in the usual sense [GM82]. However, both the "functionality" and "virtual black box" properties of obfuscators become more complex for probabilistic algorithms, so in this work, we restrict our attention to obfuscating deterministic algorithms. This restriction only makes our main (impossibility) result stronger.

Transforming Private-Key Encryption into Public-Key Encryption. Obfuscation can also be used to create new public-key encryption schemes by obfuscating a private-key encryption scheme. Given a secret key K of a private-key encryption scheme, one can publish an obfuscation of the encryption algorithm Enc_K.[3] This allows everyone to encrypt, yet only one possessing the secret key K should be able to decrypt.

Interestingly, in the original paper of Diffie and Hellman [DH76], the above was the reason given to believe that public-key cryptosystems might exist even though there were no candidates known yet. That is, they suggested that it might be possible to obfuscate a private-key encryption scheme.[4]

3.1.2 Our Results

The Basic Impossibility Result. Most of the above applications rely on the intuition that an obfuscated program is a "virtual black box." That is, anything one can efficiently compute from the obfuscated program, one should be able to efficiently compute given just oracle access to the program.

Our main result shows that it is impossible to achieve this notion of obfuscation. We prove this by constructing (from any one-way function) a family $\mathcal{F}$ of functions that is *unobfuscatable* in the sense that there is some property $\pi : \mathcal{F} \to \{0, 1\}$ such that:

- Given *any* program (circuit) that computes a function $f \in \mathcal{F}$, the value $\pi(f)$ can be efficiently computed;
- Yet, given oracle access to a (randomly selected) function $f \in \mathcal{F}$, no efficient algorithm can compute $\pi(f)$ much better than by random guessing.

Thus, there is no way of obfuscating the programs that compute these functions, even if (a) the obfuscation is meant to hide only one bit of information about the function (namely $\pi(f)$), and (b) the obfuscator itself has unbounded computation time. In fact, we show that for these functions, it is possible to efficiently reconstruct a canonical program for computing f from *every* program that computes f. This means that an adversary can completely recover the source code for f given any supposedly obfuscated program for computing f (see Section 3.4.1).

We believe that the existence of such functions shows that the "virtual black box" paradigm for obfuscators is inherently flawed. Any hope for positive results about obfuscator-like objects must abandon this viewpoint, or at least be reconciled with the existence of functions as above.

[3]This application involves the same subtlety pointed out in Footnote 2. Thus, our results regarding the (un)obfuscatability of private-key encryption schemes (described later) refer to a relaxed notion of security in which multiple encryptions of the same message are not allowed (which is consistent with a deterministic encryption algorithm).

[4]From [DH76]: "A more practical approach to finding a pair of easily computed inverse algorithms E and D; such that D is hard to infer from E, makes use of the difficulty of analyzing programs in low level languages. Anyone who has tried to determine what operation is accomplished by someone else's machine language program knows that E itself (i.e. what E does) can be hard to infer from an algorithm for E. If the program were to be made purposefully confusing through the addition of unneeded variables and statements, then determining an inverse algorithm could be made very difficult. Of course, E must be complicated enough to prevent its identification from input-output pairs. Essentially what is required is a one-way compiler: one which takes an easily understood program written in a high level language and translates it into an incomprehensible program in some machine language. The compiler is one-way because it must be feasible to do the compilation, but infeasible to reverse the process. Since efficiency in size of program and run time are not crucial in this application, such compilers may be possible if the structure of the machine language can be optimized to assist in the confusion."

Approximate Obfuscators. The basic impossibility result as described above applies to obfuscators $\mathcal{O}$ for which we require that the obfuscated program $\mathcal{O}(P)$ computes exactly the same function as the original program P. However, for some applications it may suffice that, for every input x, the programs $\mathcal{O}(P)$ and P agree on x with high probability (over the coin tosses of $\mathcal{O}$). Using some additional ideas, our impossibility result extends to such *approximate obfuscators*.

Impossibility of Applications. To give further evidence that our impossibility result is not an artifact of definitional choices, but rather that there is something inherently flawed in the "virtual black box" idea, we also demonstrate that several of the applications of obfuscators are also impossible. We do this by constructing *unobfuscatable* signature schemes, encryption schemes, and pseudorandom functions. These are objects satisfying the standard definitions of security (except for the subtlety noted in Footnote 3), but for which one can efficiently compute the secret key K from *any program* that signs (or encrypts or evaluates the pseudorandom function, resp.) relative to K. (Hence handing out "obfuscated forms" of these keyed-algorithms is highly insecure.)

In particular, we complement Canetti et. al.'s critique of the Random Oracle Methodology [CGH98]. They show that there exist (contrived) protocols that are secure in the idealized Random Oracle Model (of [BR93]), but are *insecure* when the random oracle is replaced with *any* (efficiently computable) function. Our results imply that for even for *natural* protocols that are secure in the random oracle model (e.g., Fiat-Shamir type schemes [FS86]), there exist (contrived) pseudorandom functions, such that these protocols are insecure when the random oracle is replaced with *any program* that computes the (contrived) pseudorandom function.

Obfuscating restricted complexity classes. Even though obfuscation of general programs/circuits is impossible, one may hope that it is possible to obfuscate more restricted classes of computations. However, using the pseudorandom functions of [NR97] in our construction, we can show that the impossibility result holds even when the input program P is a constant-depth threshold circuit (i.e., is in $\mathbf{TC_0}$), under widely believed complexity assumptions (e.g., the hardness of factoring).

3.1.3 Discussion : What do these Results Mean?

A natural criticism of this work is that we do not prove that any obfuscator will fail on *all* programs. For example, Van Oorschot [VO03] presents the following critique of our results:

> "While on the surface this result is quite negative for practitioners interested in software obfuscation, upon deeper inspection this is not so ... the results simply arise from the choice of definitions, model and question posed.
>
> In practice, the non-existence of such a virtual black-box generator would appear to be of little concern. Of greater interest are several different questions such as: to what proportions of programs of practical interest does this result apply; do obfuscators exist which are capable of obfuscating programs of practical interest; and can a more practical model be defined, allowing some level of non-critical information to leak from the execution of a program, provided it is not useful information to the attacker."

We now discuss the meaning of our results, and why we disagree with the above criticism and *do* think that our results have implications for practitioners interested in obfuscation. Our first point is that, although we focus on a particular definition in this chapter, it seems that any definition for a practical obfuscator $\mathcal{O}$ should satisfy the following properties:

Functionality $\mathcal{O}(P)$ computes the same function as P.[5]

Efficiency $\mathcal{O}(P)$ is at most polynomially slower than P. (Note that in practice, one would want $\mathcal{O}$ to satisfy an even stronger efficiency requirement - often even an overhead of a constant multiplicative factor is too slow.)

Hiding There should be at least some *partial* information about P that is hidden by $\mathcal{O}(P)$, and this should hold for all the programs that are contained in some class $\mathcal{C}$ of programs. In particular, at the very least it should be the case that for all the programs P in the class $\mathcal{C}$, it is hard to recover P from $\mathcal{O}(P)$. (Note that recovering P means that an adversary is able to completely reverse-engineer $\mathcal{O}(P)$ and obtain the entire source code.)

Why do practitioners need a security definition? One may argue that practitioners in obfuscation do not need any sort of definition. Rather, their job is to come up with candidate constructions which seem to be hard to break in practice. There is a problem with that approach: it is one thing to say that practitioners do not need mathematical *proofs* for the security of their constructions. It is a completely different matter to say that practitioners do not need a clearly stated *conjecture*. If someone is going to be using a particular candidate obfuscator $\mathcal{O}$ in a setting where the security of $\mathcal{O}$ is relied upon, then it seems that there should be at least a conjecture that $\mathcal{O}$ satisfies some sort of a security definition (which is sufficient for the intended use), not to mention some evidence that this conjecture is true.[6]

Thus, the question about the existence of practical obfuscators becomes the question of the existence of some algorithm $\mathcal{O}$ that satisfies the conditions above. In particular, the result of this chapter imply in order for practical obfuscators to exist there must exist an (efficiently decidable) class $\mathcal{C}$ of programs such that:

1. $\mathcal{C}$ does not contain the counterexamples presented in this chapter.

2. $\mathcal{C}$ does contain a large fraction of the programs people wish to obfuscate in practice.

However, we are not aware of any natural candidates for such a class $\mathcal{C}$. In particular, we note that the programs people wish to obfuscate in practice include hash functions and pseudorandom functions, whereas we present an example of unobfuscatable pseudorandom functions in this chapter. This means that this class $\mathcal{C}$ should be able to distinguish between a program that computes a "practical" pseudorandom function, and a program that computes "our" pseudorandom function, even though that by virtue of their pseudorandomness, these two programs have an indistinguishable input/output behavior.

It is important to note that in many cases, cryptographic applications that "seem" to require obfuscation can actually be obtained using other means. Perhaps the best example for this is a public-key encryption scheme, which Diffie and Hellman suggested to construct using obfuscation (see Footnote 4), but later were obtained using completely different methods by Rivest, Shamir

[5]As we remark in the paper, one can think of meaningful obfuscators that do not satisfy this. However, it is still a natural requirement. We also note that all the practical candidates for obfuscators we are aware of satisfy this definition.

[6]It may be sometimes possible to use a candidate obfuscator $\mathcal{O}$ in a setting where the security of $\mathcal{O}$ is *not* relied upon. For example, in some cases it may be possible to detect if $\mathcal{O}$ has been broken, and control the amount of damage this break causes. In such cases it may indeed be reasonable to use obfuscators even without having a clearly stated conjecture about their security properties.

and Adelman [RSA78]. It is of course possible to view the construction of [RSA78] as a "special purpose" obfuscation (for one particular function). However, when we discuss obfuscation in this chapter, we refer to "general purpose" obfuscators (i.e., obfuscators that work for a large class of functions).

Note: More discussion on the meaning of these results (albeit on an informal level) can be found on the author's web page on `http://www.math.ias.edu/~boaz/Papers/obf_informal.html` .

3.1.4 Additional Related Work

There are a number of heuristic approaches to obfuscation and software watermarking in the literature, as described in the surveys of Collberg and Thomborson [CT00] and Van Oorschot [VO03] and the references therein. A theoretical study of software protection was previously conducted by Goldreich and Ostrovsky [GO96], who considered *hardware-based* solutions.

Hada [Had00] gave some definitions for code obfuscators which are stronger than the definitions we consider in this paper, and showed some implications of the existence of such obfuscators. (Our result rules out also the existence of obfuscators according to the definitions of [Had00].) The results of Chapters 4 and 5 of this thesis also imply that such stronger types of obfuscators do not exist.

We note that the paper [BGI$^+$01] contains some additional results and open questions that are not included in this chapter. These include discussion of several notions that related to obfuscators, such as sampling obfuscators and watermarking schemes.

3.1.5 Organization

In Section 3.2, we give some basic definitions along with (very weak) definitions of obfuscators. In Section 3.3, we prove the impossibility of obfuscators by constructing an unobfuscatable function ensemble. In Section 3.4, we give a number of extensions of our impossibility result, including impossibility results for obfuscators which only need to approximately preserve functionality, for obfuscators computable in low circuit classes, and for some of the applications of obfuscators. We also show that our main impossibility result does not relativize.

3.2 Definitions

3.2.1 Preliminaries

We use *TM* as shorthand for Turing machine, and *PPT* as shorthand for probabilistic polynomial-time Turing machine.

If M is a TM then we denote by $\langle M\rangle$ the function $\langle M\rangle : 1^* \times \{0,1\}^* \to \{0,1\}^*$ given by:

$$\langle M\rangle(1^t, x) \stackrel{def}{=} \begin{cases} y & M(x) \text{ halts with output } y \text{ after at most } t \text{ steps} \\ \perp & \text{otherwise} \end{cases}$$

That is, oracle access to the function $\langle M\rangle$ allows a party to invoke M on any input for t steps, where t should be given in unary notation. This ensure that access to such an oracle is not more powerful than access to the description of the TM M.

3.2.2 Obfuscators

In this section, we aim to formalize the notion of obfuscators based on the "virtual black box" property as described in the introduction. Recall that this property requires that "anything that an adversary can compute from an obfuscation $\mathcal{O}(P)$ of a program P, it could also compute given just oracle access to P." We shall define what it means for the adversary to successfully compute something in this setting, and there are several choices for this (in decreasing order of generality):

(Computational indistinguishability) The most general choice is not to restrict the nature of what the adversary is trying to compute, and merely require that it is possible, given just oracle access to P, to produce an output distribution that is computationally indistinguishable from what the adversary computes when given $\mathcal{O}(P)$.

(Satisfying a relation) An alternative is to consider the adversary as trying to produce an output that satisfies an arbitrary (possibly polynomial-time) relation with the original program P, and require that it is possible, given just oracle access to P, to succeed with roughly the same probability as the adversary does when given $\mathcal{O}(P)$.

(Computing a function) A weaker requirement is to restrict the previous requirement to relations which are functions; that is, the adversary is trying to compute some function of the original program.

(Computing a predicate) The weakest is to restrict the previous requirement to $\{0,1\}$-valued functions; that is, the adversary is trying to decide some property of the original program.

Since we will be proving impossibility results, our results are strongest when we adopt the weakest requirement (i.e., the last one). This yields two definitions for obfuscators, one for programs defined by Turing machines and one for programs defined by circuits.

Definition 3.2.1 (TM obfuscator). A probabilistic algorithm $\mathcal{O}$ is a *TM obfuscator* if the following three conditions hold:

Functionality: For every TM M, the string $\mathcal{O}(M)$ describes a TM that computes the same function as M.

Polynomial slowdown: The description length and running time of $\mathcal{O}(M)$ are at most polynomially larger than that of M. That is, there is a polynomial p such that for every TM M, $|\mathcal{O}(M)| \leq p(|M|)$, and if M halts in t steps on some input x, then $\mathcal{O}(M)$ halts within $p(t)$ steps on x.

"Virtual black box" property: For any PPT A, there is a PPT S such that for all TMs M

$$\left|\Pr\left[A(\mathcal{O}(M)) = 1\right] - \Pr\left[S^{\langle M \rangle}(1^{|M|}) = 1\right]\right| \leq \mathsf{neg}(|M|)$$

.

We say that $\mathcal{O}$ is *efficient* if it runs in polynomial time.

Definition 3.2.2 (circuit obfuscator). A probabilistic algorithm $\mathcal{O}$ is a *(circuit) obfuscator* if the following three conditions hold:

Functionality: For every circuit C, the string $\mathcal{O}(C)$ describes a circuit that computes the same function as C.

Polynomial slowdown: There is a polynomial p such that for every circuit C, $|\mathcal{O}(C)| \leq p(|C|)$.

"Virtual black box" property: For any PPT A, there is a PPT S such that for all circuits C

$$\left|\Pr\left[A(\mathcal{O}(C)) = 1\right] - \Pr\left[S^C(1^{|C|}) = 1\right]\right| \leq \mathsf{neg}(|C|)$$

.

We say that $\mathcal{O}$ is *efficient* if it runs in polynomial time.

We call the first two requirements (functionality and polynomial slowdown) the *syntactic requirements* of obfuscation, as they do not address the issue of security at all.

There are a couple of other natural formulations of the "virtual black box" property. The first, which more closely follows the informal discussion above, asks that for every predicate π, the probability that $A(\mathcal{O}(C)) = \pi(C)$ is at most the probability that $S^C(1^{|C|}) = \pi(C)$ plus a negligible term. It is easy to see that this requirement is equivalent to the one above. Another formulation refers to the distinguishability between obfuscations of two TMs/circuits: ask that for every C_1 and C_2, $|\Pr\left[A(\mathcal{O}(C_1)) = 1\right] - \Pr\left[A(\mathcal{O}(C_2))\right]|$ is approximately equal to $|\Pr\left[S^{C_1}(1^{|C_1|}, 1^{|C_2|}) = 1\right] - \Pr\left[S^{C_2}(1^{|C_1|}, 1^{|C_2|})\right]|$. This definition appears to be slightly weaker than the ones above, but our impossibility proof also rules it out.

Note that in both definitions, we have chosen to simplify the definition by using the size of the TM/circuit to be obfuscated as a security parameter. One can always increase this length by padding to obtain higher security.

The main difference between the circuit and TM obfuscators is that a circuit computes a function with finite domain (all the inputs of a particular length) while a TM computes a function with infinite domain. Note that if we had not restricted the size of the obfuscated circuit $\mathcal{O}(C)$, then the (exponential size) list of all the values of the circuit would be a valid obfuscation (provided we allow S running time $\mathrm{poly}(|\mathcal{O}(C)|)$ rather than $\mathrm{poly}(|C|)$). For Turing machines, it is not clear how to construct such an obfuscation, even if we are allowed an exponential slowdown. Hence obfuscating TMs is intuitively harder. Indeed, it is relatively easy to prove:

Proposition 3.2.3. *If a TM obfuscator exists, then a circuit obfuscator exists.*

Thus, when we prove our impossibility result for circuit obfuscators, the impossibility of TM obfuscators will follow. However, considering TM obfuscators will be useful as motivation for the proof.

We note that, from the perspective of applications, Definitions 3.2.1 and 3.2.2 are already too weak to have the wide applicability discussed in the introduction. The point is that they are nevertheless *impossible* to satisfy (as we will prove).

3.3 The Main Impossibility Result

To state our main result we introduce the notion of unobfuscatable function ensemble.

Definition 3.3.1. An *unobfuscatable function ensemble* is an ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ of distributions $\mathcal{H}_k$ on finite functions (from, say, $\{0,1\}^{l_{\mathrm{in}}(k)}$ to $\{0,1\}^{l_{\mathrm{out}}(k)}$) satisfying:

Efficient computability: Every function $f \leftarrow_{\mathrm{R}} \mathcal{H}_k$ is computable by a circuit of size $\mathrm{poly}(k)$. (Moreover, a distribution on circuits consistent with $\mathcal{H}_k$ can be sampled uniformly in time $\mathrm{poly}(k)$.)

Unobfuscatability: There exists a function $\pi : \bigcup_{k \in \mathbb{N}} \mathrm{Supp}(\mathcal{H}_k) \to \{0,1\}$ such that

1. $\pi(f)$ is hard to compute with black-box access to f: For any PPT S

$$\Pr_{f \leftarrow_{\mathrm{R}} \mathcal{H}_k}[S^f(1^k) = \pi(f)] \leq \frac{1}{2} + \mathsf{neg}(k)$$

2. $\pi(f)$ is easy to compute with access to any circuit that computes f: There exists a PPT A such that for any $f \in \bigcup_{k \in \mathbb{N}} \mathrm{Supp}(\mathcal{H}_k)$ and for any circuit C that computes f

$$A(C) = \pi(f)$$

Connection with learning. Note that an unobfuscatable function ensemble must correspond to a concept class that is hard to exactly-learn with queries. Indeed, if given oracle access to any $f \in \mathcal{H}$ one can efficiently find a circuit computing f, then items 1 and 2 in the unobfuscatability condition would contradict each other.

We prove in Theorem 3.3.11 that, assuming one-way functions exist, there exists an unobfuscatable function ensemble. This implies that, under the same assumption, there is no obfuscator that satisfies Definition 3.2.2 (actually we prove the latter fact directly in Theorem 3.3.8). Since the existence of an *efficient* obfuscator implies the existence of one-way functions (Lemma 3.3.9), we conclude that efficient obfuscators do not exist (unconditionally).

However, the existence of unobfuscatable function ensemble has even stronger implications. As mentioned in the introduction, these functions can not be obfuscated even if we allow the following relaxations to the obfuscator:

1. As mentioned above, the obfuscator does not have to run in polynomial time — it can be any random process.

2. The obfuscator has only to work for functions in $\mathrm{Supp}(\mathcal{H}_k)$ and only for a non-negligible fraction of these functions under the distributions $\mathcal{H}_k$.

3. The obfuscator has only to hide an *a priori* fixed property π from an *a priori* fixed adversary A.

Structure of the Proof of the Main Impossibility Result. We shall prove our result by first defining obfuscators that are secure also when applied to several (e.g., two) algorithms and proving that they do not exist. Then we shall modify the construction in this proof to prove that TM obfuscators in the sense of Definition 3.2.1 do not exist. After that, using an additional construction (which requires one-way functions), we will prove that a circuit obfuscator as defined in Definition 3.2.2 does not exist if one-way functions exist. We will then observe that our proof actually yields an unobfuscatable function ensemble (Theorem 3.3.11).

3.3.1 Obfuscating two TMs/circuits

Obfuscators as defined in the previous section provide a "virtual black box" property when a single program is obfuscated, but the definitions do not say anything about what happens when the adversary can inspect more than one obfuscated program. In this section, we will consider extensions of those definitions to obfuscating two programs, and prove that they are impossible to meet. The proofs will provide useful motivation for the impossibility of the original one-program definitions.

Definition 3.3.2 (2-TM obfuscator). A *2-TM obfuscator* is defined in the same way as a TM obfuscator, except that the "virtual black box" property is strengthened as follows:

"Virtual black box" property: For any PPT A, there is a PPT S such that for all TMs M, N

$$\left|\Pr\Big[A(\mathcal{O}(M), \mathcal{O}(N)) = 1\Big] - \Pr\Big[S^{\langle M\rangle,\langle N\rangle}(1^{|M|+|N|}) = 1\Big]\right| \leq \mathsf{neg}(\min\{|M|, |N|\})$$

Definition 3.3.3 (2-circuit obfuscator). A *2-circuit obfuscator* is defined in the same way as a circuit obfuscator, except that the "virtual black box" property is replaced with the following:

"Virtual black box" property: For any PPT A, there is a PPT S such that for all circuits C, D

$$\left|\Pr\Big[A(\mathcal{O}(C), \mathcal{O}(D)) = 1\Big] - \Pr\Big[S^{C,D}(1^{|C|+|D|}) = 1\Big]\right| \leq \mathsf{neg}(\min\{|C|, |D|\})$$

Proposition 3.3.4. *Neither 2-TM nor 2-circuit obfuscators exist.*

Proof. We begin by showing that 2-TM obfuscators do not exist. Suppose, for sake of contradiction, that there exists a 2-TM obfuscator $\mathcal{O}$. The essence of this proof, and in fact of all the impossibility proofs in this paper, is that there is a fundamental difference between getting black-box access to a function and getting a program that computes it, no matter how obfuscated: A program is a succinct description of the function, on which one can perform computations (or run other programs). Of course, if the function is (exactly) learnable via oracle queries (i.e., one can acquire a program that computes the function by querying it at a few locations), then this difference disappears. Hence, to get our counterexample, we will use a function that cannot be exactly learned with oracle queries. A very simple example of such an unlearnable function follows. For strings $\alpha, \beta \in \{0,1\}^k$, define the Turing machine

$$C_{\alpha,\beta}(x) \stackrel{def}{=} \begin{cases} \beta & x = \alpha \\ 0^k & \text{otherwise} \end{cases}$$

We assume that on input x, $C_{\alpha,\beta}$ runs in $10 \cdot |x|$ steps (the constant 10 is arbitrary). Now we will define a TM $D_{\alpha,\beta}$ that, given the code of a TM C, can distinguish between the case that C computes the same function as $C_{\alpha,\beta}$ from the case that C computes the same function as $C_{\alpha',\beta'}$ for any $(\alpha', \beta') \neq (\alpha, \beta)$.

$$D_{\alpha,\beta}(C) \stackrel{def}{=} \begin{cases} 1 & C(\alpha) = \beta \\ 0 & \text{otherwise} \end{cases}$$

(Actually, this function is uncomputable. However, as we shall see below, we can use a modified version of $D_{\alpha,\beta}$ that only considers the execution of $C(\alpha)$ for $\mathrm{poly}(k)$ steps, and outputs 0 if C does not halt within that many steps, for some fixed polynomial $\mathrm{poly}(\cdot)$. We will ignore this issue for now, and elaborate on it later.) Note that $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$ have description size $\Theta(k)$.

Consider an adversary A, which, given two (obfuscated) TMs as input, simply runs the second TM on the first one. That is, $A(C, D) = D(C)$. (Actually, like we modified $D_{\alpha,\beta}$ above, we also will modify A to only run D on C for $\mathrm{poly}(|C|, |D|)$ steps, and output 0 if D does not halt in that time.) Thus, for any $\alpha, \beta \in \{0,1\}^k$,

$$\Pr\left[A(\mathcal{O}(C_{\alpha,\beta}), \mathcal{O}(D_{\alpha,\beta})) = 1\right] = 1 \tag{3.1}$$

Observe that any poly(k)-time algorithm S which has oracle access to $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$ has only exponentially small probability (for a random α and β) of querying either oracle at a point where its value is nonzero. Hence, if we let Z_k be a Turing machine that always outputs 0^k, then for every PPT S,

$$\left|\Pr\left[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k)=1\right]-\Pr\left[S^{Z_k,D_{\alpha,\beta}}(1^k)=1\right]\right|\leq 2^{-\Omega(k)}, \tag{3.2}$$

where the probabilities are taken over α and β selected uniformly in $\{0,1\}^k$ and the coin tosses of S. On the other hand, by the definition of A we have:

$$\Pr\left[A(\mathcal{O}(Z_k),\mathcal{O}(D_{\alpha,\beta}))=1\right]=0 \tag{3.3}$$

The combination of Equations (3.1), (3.2), and (3.3) contradict the fact that $\mathcal{O}$ is a 2-TM obfuscator.

In the above proof, we ignored the fact that we had to truncate the running times of A and $D_{\alpha,\beta}$. When doing so, we must make sure that Equations (3.1) and (3.3) still hold. Equation (3.1) involves executing (a) $A(\mathcal{O}(D_{\alpha,\beta}),\mathcal{O}(C_{\alpha,\beta}))$, which in turn amounts to executing (b) $\mathcal{O}(D_{\alpha,\beta})(\mathcal{O}(C_{\alpha,\beta}))$. By definition (b) has the same functionality as $D_{\alpha,\beta}(\mathcal{O}(C_{\alpha,\beta}))$, which in turn involves executing (c) $\mathcal{O}(C_{\alpha,\beta})(\alpha)$. Yet the functionality requirement of the obfuscator definition assures us that (c) has the same functionality as $C_{\alpha,\beta}(\alpha)$. By the polynomial slowdown property of obfuscators, execution (c) only takes poly$(10\cdot k)$ = poly(k) steps, which means that $D_{\alpha,\beta}(\mathcal{O}(C_{\alpha,\beta}))$ need only run for poly(k) steps. Thus, again applying the polynomial slowdown property, execution (b) takes poly(k) steps, which finally implies that A need only run for poly(k) steps. The same reasoning holds for Equation (3.3), using Z_k instead of $C_{\alpha,\beta}$.[7] Note that all the polynomials involved are *fixed* once we fix the polynomial $p(\cdot)$ of the polynomial slowdown property.

The proof for the 2-circuit case is very similar to the 2-TM case, with a related, but slightly different subtlety. Suppose, for sake of contradiction, that $\mathcal{O}$ is a 2-circuit obfuscator. For $k\in\mathbb{N}$ and $\alpha,\beta\in\{0,1\}^k$, define Z_k, $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$ in the same way as above but as circuits rather than TMs, and define an adversary A by $A(C,D)=D(C)$. (Note that the issues of A and $D_{\alpha,\beta}$'s running times go away in this setting, since circuits can always be evaluated in time polynomial in their size.) The new subtlety here is that the definition of A as $A(C,D)=D(C)$ only makes sense when the input length of D is larger than the size of C (note that one can always pad C to a larger size). Thus, for the analogues of Equations (3.1) and (3.3) to hold, the input length of $D_{\alpha,\beta}$ must be larger than the sizes of the *obfuscations* of $C_{\alpha,\beta}$ and Z_k. However, by the polynomial slowdown property of obfuscators, it suffices to let $D_{\alpha,\beta}$ have input length poly(k) and the proof works as before.

□

3.3.2 Obfuscating one TM/circuit

Our approach to extending the two-program obfuscation impossibility results to the one-program definitions is to combine the two programs constructed above into one. This will work in a quite straightforward manner for TM obfuscators, but will require new ideas for circuit obfuscators.

Combining functions and programs. For functions, TMs, or circuits $f_0,f_1:X\to Y$, define their *combination* $f_0\#f_1:\{0,1\}\times X\to Y$ by $(f_0\#f_1)(b,x)\stackrel{def}{=}f_b(x)$. Conversely, if we are given

[7] Another, even more minor subtlety that we ignored is that, strictly speaking, A only has running time polynomial in the description of the *obfuscations* of $C_{\alpha,\beta}$, $D_{\alpha,\beta}$, and Z_k, which could conceivably be shorter than the original TM descriptions. But a counting argument shows that for all but an exponentially small fraction of pairs $(\alpha,\beta)\in\{0,1\}^k\times\{0,1\}^k$, $\mathcal{O}(C_{\alpha,\beta})$ and $\mathcal{O}(D_{\alpha,\beta})$ must have description size $\Omega(k)$.

a TM (resp., circuit) $C : \{0,1\} \times X \to Y$, we can efficiently decompose C into $C_0 \# C_1$ by setting $C_b(x) \stackrel{def}{=} C(b,x)$; note that C_0 and C_1 have size and running time essentially the same as that of C. Observe that having oracle access to a combined function $f_0 \# f_1$ is equivalent to having oracle access to f_0 and f_1 individually.

Theorem 3.3.5. *TM obfuscators do not exist.*

Proof Sketch: Suppose, for sake of contradiction, that there exists a TM obfuscator $\mathcal{O}$. For $\alpha, \beta \in \{0,1\}^k$, let $C_{\alpha,\beta}$, $D_{\alpha,\beta}$, and Z_k be the TMs defined in the proof of Proposition 3.3.4. Combining these, we get the TMs $F_{\alpha,\beta} = C_{\alpha,\beta} \# D_{\alpha,\beta}$ and $G_{\alpha,\beta} = Z_k \# C_{\alpha,\beta}$.

We consider an adversary A analogous to the one in the proof of Proposition 3.3.4, augmented to first decompose the program it is fed. That is, on input a TM F, algorithm A first decomposes F into $F_0 \# F_1$ and then outputs $F_1(F_0)$. (As in the proof of Proposition 3.3.4, A actually should be modified to run in time $\mathrm{poly}(|F|)$.) Let S be the PPT simulator for A guaranteed by Definition 3.2.1. Just as in the proof of Proposition 3.3.4, we have:

$$\begin{aligned} \Pr\left[A(\mathcal{O}(F_{\alpha,\beta})) = 1\right] = 1 \text{ and } \Pr\left[A(\mathcal{O}(G_{\alpha,\beta})) = 1\right] &= 0 \\ \left|\Pr\left[S^{F_{\alpha,\beta}}(1^k) = 1\right] - \Pr\left[S^{G_{\alpha,\beta}}(1^k) = 1\right]\right| &\leq 2^{-\Omega(k)}, \end{aligned}$$

where the probabilities are taken over uniformly selected $\alpha, \beta \in \{0,1\}^k$, and the coin tosses of A, S, and $\mathcal{O}$. This contradicts Definition 3.2.1. $\square$

There is a difficulty in trying to carry out the above argument in the circuit setting. (This difficulty is related to (but more serious than) the same subtlety regarding the circuit setting discussed earlier.) In the above proof, the adversary A, on input $\mathcal{O}(F_{\alpha,\beta})$, attempts to evaluate $F_1(F_0)$, where $F_0 \# F_1 = \mathcal{O}(F_{\alpha,\beta}) = \mathcal{O}(C_{\alpha,\beta} \# D_{\alpha,\beta})$. In order for this to make sense in the circuit setting, the size of the circuit F_0 must be at most the input length of F_1 (which is the same as the input length of $D_{\alpha,\beta}$). But, since the output $F_0 \# F_1$ of the obfuscator can be polynomially larger than its input $C_{\alpha,\beta} \# D_{\alpha,\beta}$, we have no such guarantee. Furthermore, note that if we compute F_0, F_1 in the way we described above (i.e., $F_b(x) \stackrel{def}{=} \mathcal{O}(F_{\alpha,\beta})(b,x)$) then we'll have $|F_0| = |F_1|$ and so F_0 will necessarily be larger than F_1's input length.

To get around this, we modify $D_{\alpha,\beta}$ in a way that will allow A, when given $D_{\alpha,\beta}$ and a circuit C, to test whether $C(\alpha) = \beta$ even when C is larger than the input length of $D_{\alpha,\beta}$. Of course, oracle access to $D_{\alpha,\beta}$ should not reveal α and β, because we do not want the simulator S to be able to test whether $C(\alpha) = \beta$ given just oracle access to C and $D_{\alpha,\beta}$. We will construct such functions $D_{\alpha,\beta}$ based on pseudorandom functions [GGM86].

Lemma 3.3.6. *If one-way functions exist, then for every $k \in \mathbb{N}$ and $\alpha, \beta \in \{0,1\}^k$, there is a distribution $\mathcal{D}_{\alpha,\beta}$ on circuits such that:*

1. *Every $D \in \mathrm{Supp}(\mathcal{D}_{\alpha,\beta})$ is a circuit of size* $\mathrm{poly}(k)$.
2. *There is a polynomial-time algorithm A such that for any circuit C, and any $D \in \mathrm{Supp}(\mathcal{D}_{\alpha,\beta})$, $A^D(C, 1^k) = 1$ iff $C(\alpha) = \beta$.*
3. *For any PPT S, $\Pr\left[S^D(1^k) = \alpha\right] = \mathsf{neg}(k)$, where the probability is taken over $\alpha, \beta \leftarrow_R \{0,1\}^k$, $D \leftarrow_R \mathcal{D}_{\alpha,\beta}$, and the coin tosses of S.*

Proof. Basically, the construction implements a private-key "homomorphic encryption" scheme. More precisely, the functions in $\mathcal{D}_{\alpha,\beta}$ will consist of three parts. The first part gives out an encryption of the bits of α (under some private-key encryption scheme). The second part provides the ability to perform binary Boolean operations on encrypted bits, and the third part tests whether a sequence of encryptions consists of encryptions of the bits of β. These operations will enable one to efficiently test whether a given circuit C satisfies $C(\alpha) = \beta$, while keeping α and β hidden when only oracle access to C and $D_{\alpha,\beta}$ is provided.

We begin with any one-bit (probabilistic) private-key encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ that satisfies indistinguishability under *chosen plaintext* and *nonadaptive chosen ciphertext* attacks. Informally, this means that an encryption of 0 should be indistinguishable from an encryption of 1 even for adversaries that have access to encryption and decryption oracles prior to receiving the challenge ciphertext, and access to just an encryption oracle after receiving the challenge ciphertext. (See [KY00] for formal definitions.) We note that such encryptions schemes exist if one-way functions exist; indeed, the "standard" encryption scheme $\mathrm{Enc}_K(b) = (r, f_K(r) \oplus b)$, where $r \leftarrow_{\mathrm{R}} \{0,1\}^{|K|}$ and f_K is a pseudorandom function, has this property.

Now we consider a "homomorphic encryption" algorithm Hom, which takes as input a private-key K and two ciphertexts c and d (w.r.t. this key K), and a binary boolean operation $\odot$ (specified by its 2×2 truth table). We define

$$\mathrm{Hom}_K(c, d, \odot) \stackrel{def}{=} \mathrm{Enc}_K(\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d)).$$

It can be shown that such an encryption scheme retains its security even if the adversary is given access to a Hom oracle. This is formalized in the following claim:

Claim 3.3.7. *For every PPT A,*

$$\left|\Pr\left[A^{\mathrm{Hom}_K,\mathrm{Enc}_K}(\mathrm{Enc}_K(0)) = 1\right] - \Pr\left[A^{\mathrm{Hom}_K,\mathrm{Enc}_K}(\mathrm{Enc}_K(1)) = 1\right]\right| \le \mathit{neg}(k).$$

Proof. Suppose there were a PPT A violating the claim. First, we argue that we can replace the responses to all of A'S Hom_K-oracle queries with encryptions of 0 with only a negligible effect on A's distinguishing gap. This follows from indistinguishability under chosen plaintext and ciphertext attacks and a hybrid argument: Consider hybrids where the first i oracle queries are answered according to Hom_K and the rest with encryptions of 0. Any advantage in distinguishing two adjacent hybrids must be due to distinguishing an encryption of 1 from an encryption of 0. The resulting distinguisher can be implemented using oracle access to encryption and decryption oracles prior to receiving the challenge ciphertext (and an encryption oracle afterward).

Once we have replaced the Hom_K-oracle responses with encryptions of 0, we have an adversary that can distinguish an encryption of 0 from an encryption of 1 when given access to just an encryption oracle. This contradicts indistinguishability under chosen plaintext attack. □

Now we return to the construction of our circuit family $\mathcal{D}_{\alpha,\beta}$. For a key K, let $E_{K,\alpha}$ be an algorithm which, on input i outputs $\mathrm{Enc}_K(\alpha_i)$, where α_i is the i'th bit of α. Let $B_{K,\beta}$ be an algorithm which when fed a k-tuple of ciphertexts $(c_1, \ldots, c_k)$ outputs 1 if for all i, $\mathrm{Dec}_K(c_i) = \beta_i$, where $\beta_1, \ldots, \beta_k$ are the bits of β. A random circuit from $\mathcal{D}_{\alpha,\beta}$ will essentially be the algorithm

$$D_{K,\alpha,\beta} \stackrel{def}{=} E_{K,\alpha} \# \mathrm{Hom}_K \# B_{K,\beta}$$

(for a uniformly selected key K). One minor complication is that $D_{K,\alpha,\beta}$ is actually a *probabilistic* algorithm, since $E_{K,\alpha}$ and Hom_K employ probabilistic encryption, whereas the lemma requires deterministic functions. This can be solved in the usual way, by using pseudorandom functions. Let $q = q(k)$ be the input length of $D_{K,\alpha,\beta}$ and $m = m(k)$ the maximum number of random bits used by $D_{K,\alpha,\beta}$ on any input. We can select a pseudorandom function $f_{K'} : \{0,1\}^q \to \{0,1\}^m$, and let $D'_{K,\alpha,\beta,K'}$ be the (deterministic) algorithm, which on input $x \in \{0,1\}^q$ evaluates $D_{K,\alpha,\beta}(x)$ using randomness $f_{K'}(x)$.

Define the distribution $\mathcal{D}_{\alpha,\beta}$ to be $D'_{K,\alpha,\beta,K'}$, over uniformly selected keys K and K'. We argue that this distribution has the properties stated in the lemma. By construction, each $D'_{K,\alpha,\beta,K'}$ is computable by circuit of size $\mathrm{poly}(k)$, so Property 1 is satisfied.

For Property 2, consider an algorithm A that on input C and oracle access to $D'_{K,\alpha,\beta,K'}$ (which, as usual, we can view as access to (deterministic versions of) the three separate oracles $E_{K,\alpha}$, Hom_K, and $B_{K,\alpha}$), proceeds as follows: First, with k oracle queries to the $E_{K,\alpha}$ oracle, A obtains encryptions of each of the bits of α. Then, A uses the Hom_K oracle to do a gate-by-gate emulation of the computation of $C(\alpha)$, in which A obtains encryptions of the values at each gate of C. In particular, A obtains encryptions of the values at each output gate of C (on input α). It then feeds these output encryptions to $D_{K,\beta}$, and outputs the response to this oracle query. By construction, A outputs 1 iff $C(\alpha) = \beta$.

Finally, we verify Property 3. Let S be any PPT algorithm. We must show that S has only a negligible probability of outputting α when given oracle access to $D'_{K,\alpha,\beta,K'}$ (over the choice of K, α, β, K', and the coin tosses of S). By the pseudorandomness of $f_{K'}$, we can replace oracle access to the function $D'_{K,\alpha,\beta,K'}$ with oracle access to the probabilistic algorithm $D_{K,\alpha,\beta}$ with only a negligible effect on S's success probability. Oracle access to $D_{K,\alpha,\beta}$ is equivalent to oracle access to $E_{K,\alpha}$, Hom_K, and $B_{K,\beta}$. Since β is independent of α and K, the probability that S queries $B_{K,\beta}$ at a point where its value is nonzero (i.e., at a sequence of encryptions of the bits of β) is exponentially small, so we can remove S's queries to $B_{K,\beta}$ with only a negligible effect on the success probability. Oracle access to $E_{K,\alpha}$ is equivalent to giving S polynomially many encryptions of each of the bits of α. Thus, we must argue that S cannot compute α with nonnegligible probability from these encryptions and oracle access to Hom_K. This follows from the fact that the encryption scheme remains secure in the presence of a Hom_K oracle (Claim 3.3.7) and a hybrid argument. □

Now we can prove the impossibility of circuit obfuscators.

Theorem 3.3.8. *If one-way functions exist, then circuit obfuscators do not exist.*

Proof. Suppose, for sake of contradiction, that there exists a circuit obfuscator $\mathcal{O}$. For $k \in \mathbb{N}$ and $\alpha, \beta \in \{0,1\}^k$, let Z_k and $C_{\alpha,\beta}$ be the circuits defined in the proof of Proposition 3.3.4, and let $\mathcal{D}_{\alpha,\beta}$ be the distribution on circuits given by Lemma 3.3.6. For each $k \in \mathbb{N}$, consider the following two distributions on circuits of size $\mathrm{poly}(k)$:

$\mathcal{F}_k$**:** Choose α and β uniformly in $\{0,1\}^k$, $D \xleftarrow{}_{\mathrm{R}} \mathcal{D}_{\alpha,\beta}$. Output $C_{\alpha,\beta}\#D$.

$\mathcal{G}_k$**:** Choose α and β uniformly in $\{0,1\}^k$, $D \xleftarrow{}_{\mathrm{R}} \mathcal{D}_{\alpha,\beta}$. Output $Z_k\#D$.

Let A be the PPT algorithm guaranteed by Property 2 in Lemma 3.3.6, and consider a PPT A' which, on input a circuit F, decomposes $F = F_0\#F_1$ and evaluates $A^{F_1}(F_0, 1^k)$, where k is the input length of F_0. Thus, when fed a circuit from $\mathcal{O}(\mathcal{F}_k)$ (resp., $\mathcal{O}(\mathcal{G}_k)$), A' is evaluating $A^D(C, 1^k)$ where D computes the same function as some circuit from $\mathcal{D}_{\alpha,\beta}$ and C computes the same function as $C_{\alpha,\beta}$ (resp., Z_k). Therefore, by Property 2 in Lemma 3.3.6, we have:

We now argue that for any PPT algorithm S

$$\left|\Pr\left[S^{\mathcal{F}_k}(1^k) = 1\right] - \Pr\left[S^{\mathcal{G}_k}(1^k) = 1\right]\right| \leq 2^{-\Omega(k)},$$

which will contradict the definition of circuit obfuscators. Having oracle access to a circuit from $\mathcal{F}_k$ (respectively, $\mathcal{G}_k$) is equivalent to having oracle access to $C_{\alpha,\beta}$ (resp., Z_k) and $D \leftarrow_{\mathrm{R}} \mathcal{D}_{\alpha,\beta}$, where α, β are selected uniformly in $\{0,1\}^k$. Property 3 of Lemma 3.3.6 implies that the probability that S queries the first oracle at α is negligible, and hence S cannot distinguish that oracle being $C_{\alpha,\beta}$ from it being Z_k. □

We can remove the assumption that one-way functions exist for *efficient* circuit obfuscators via the following (easy) lemma.

Lemma 3.3.9. *If efficient obfuscators exist, then one-way functions exist.*

Proof Sketch: Suppose that $\mathcal{O}$ is an efficient obfuscator as per Definition 3.2.2. For $\alpha \in \{0,1\}^k$ and $b \in \{0,1\}$, let $C_{\alpha,b} : \{0,1\}^k \to \{0,1\}$ be the circuit defined by

$$C_{\alpha,b}(x) \stackrel{def}{=} \begin{cases} b & x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

Now define $f_k(\alpha, b, r) \stackrel{def}{=} \mathcal{O}(C_{\alpha,b}; r)$, i.e. the obfuscation of $C_{\alpha,b}$ using coin tosses r. We will argue that $f = \bigcup_{k\in\mathbb{N}} f_k$ is a one-way function. Clearly f_k can be evaluated in time $\mathrm{poly}(k)$. Since the bit b is information-theoretically determined by $f_k(\alpha, b, r)$, to show that f is one-way it suffices to show that b is a *hard-core bit* of f. To prove this, we first observe that for any PPT S,

$$\Pr_{\alpha,b}\left[S^{C_{\alpha,b}}(1^k) = b\right] \leq \frac{1}{2} + \mathsf{neg}(k).$$

By the virtual black box property of $\mathcal{O}$, it follows that for any PPT A,

$$\Pr_{\alpha,b,r}\left[A(f(\alpha, b, r)) = b\right] = \Pr_{\alpha,b,r}\left[A(\mathcal{O}(C_{\alpha,b}; r)) = b\right] \leq \frac{1}{2} + \mathsf{neg}(k).$$

This demonstrates that b is indeed a hard-core bit of f, and hence that f is one-way. □

Corollary 3.3.10. *Efficient circuit obfuscators do not exist (unconditionally).*

As stated above, our impossibility proof can be cast in terms of "unobfuscatable functions":

Theorem 3.3.11 (unobfuscatable functions). *If one-way functions exist, then there exists an unobfuscatable function ensemble.*

Proof. Let $\mathcal{F}_k$ and $\mathcal{G}_k$ be the distributions on functions in the proof of Theorem 3.3.8,and let $\mathcal{H}_k$ be the distribution that, with probability 1/2 outputs a sample of $\mathcal{F}_k$ and with probability 1/2 outputs a sample of $\mathcal{G}_k$. We claim that $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ is an unobfuscatable function ensemble.

The fact that $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ is efficiently computable is obvious. We define $\pi(f)$ to be 1 if $f \in \bigcup_k \mathrm{Supp}(\mathcal{F}_k)$ and 0 otherwise (note that $(\bigcup_k \mathrm{Supp}(\mathcal{F}_k)) \cap (\bigcup_k \mathrm{Supp}(\mathcal{G}_k)) = \emptyset$ and so $\pi(f) = 0$ for any $f \in \bigcup_k \mathrm{Supp}(\mathcal{G}_k)$). The algorithm A' given in the proof of Theorem 3.3.8 shows that $\pi(f)$ can be computed in polynomial time from any circuit computing $f \in \mathrm{Supp}(\mathcal{H}_k)$. Because oracle access to $\mathcal{F}_k$ cannot be distinguished from oracle access to $\mathcal{G}_k$ (as shown in the proof of Theorem 3.3.8), it follows that $\pi(f)$ cannot be computed from an oracle for $f \leftarrow_{\mathrm{R}} \mathcal{H}_k$ with probability noticeably greater than 1/2. □

3.4 Extensions

3.4.1 Totally unobfuscatable functions

Some of the extensions of our impossibility result require a somewhat stronger form of unobfuscatable functions, in which it is not only possible to compute $\pi(f)$ from any circuit for f, but even to recover the "original" circuit for f. This can be achieved by a slight modification of our construction. It will also be useful to extend the construction so that not only the one bit $\pi(f)$ is unpredictable given oracle access to f, but rather that there are many bits of information about f which are completely pseudorandom. These properties are captured by the definition below. In this definition, it will be convenient to identify the functions f in our family with the canonical circuits that compute them.

Definition 3.4.1 (Totally unobfuscatable functions). A *totally unobfuscatable function ensemble* is an ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ of distributions $\mathcal{H}_k$ on *circuits* (from, say, $\{0,1\}^{l_{\mathrm{in}}(k)}$ to $\{0,1\}^{l_{\mathrm{out}}(k)}$) satisfying:

Efficient computability: Every circuit $f \in \mathrm{Supp}(\mathcal{H}_k)$ is of size $\mathrm{poly}(k)$. Moreover, $f \xleftarrow{\mathrm{R}} \mathrm{Supp}(\mathcal{H}_k)$ can be sampled uniformly in time $\mathrm{poly}(k)$.

Unobfuscatability: There exists a poly-time computable function $\pi : \bigcup_{k\in\mathbb{N}} \mathrm{Supp}(\mathcal{H}_k) \to \{0,1\}^*$, such that

1. $\pi(f)$ is pseudorandom given black-box access to f: For any PPT S

$$\left| \Pr_{f \xleftarrow{\mathrm{R}} \mathcal{H}_k}[S^f(\pi(f)) = 1] - \Pr_{f \xleftarrow{\mathrm{R}} \mathcal{H}_k, z \xleftarrow{\mathrm{R}} \{0,1\}^k}[S^f(z) = 1] \right| \le \mathsf{neg}(k)$$

2. f is easy to reconstruct given any other circuit for f: There exists a PPT A such that for any $f \in \bigcup_k \mathrm{Supp}(\mathcal{H}_k)$ and for any circuit C that computes the same function as f

$$A(C) = f$$

,

Note that totally unobfuscatable functions imply unobfuscatable functions: given oracle access to a totally unobfuscatable f, pseudorandomness implies that the first bit of $\pi(f)$ cannot be computed with probability noticeably more than $1/2$, and given any circuit for f, one can efficiently find the canonical circuit for f, from which one can compute $\pi(f)$ (and in particular, its first bit).

Theorem 3.4.2 (totally unobfuscatable functions). *If one-way functions exist, then there exists a totally unobfuscatable function ensemble.*

Proof Sketch: The first step is to observe that the ensemble $\mathcal{D}_{\alpha,\beta}$ of Lemma 3.3.6 can be modified so that Property 2 instead says $A^D(C, 1^k) = \alpha$ if $C(\alpha) = \beta$ and $A^D(C, 1^k) = 0^k$ otherwise. (To achieve this, replace $B_{K,\beta}$ with $B'_{K,\alpha,\beta}$ which outputs α when fed a sequence of ciphertexts $(c_1, \dots, c_k)$ whose decryptions are the bits of β and outputs 0^k otherwise.)

Now our totally unobfuscatable function ensemble $\mathcal{H}_k$ is defined as follows.

$\mathcal{H}_k$**:** Choose α, β, γ uniformly in $\{0,1\}^k$, $D \xleftarrow{\mathrm{R}} \mathcal{D}_{\alpha,\beta}$. Output $C_{\alpha,\beta} \# D \# C_{\alpha,(D,\gamma)}$.

(Above, $C_{\alpha,(D,\gamma)}$ is the circuit which on input α outputs (D,γ), and on all other inputs outputs $0^{|(D,\gamma)|}$.)

Efficiency is clearly satisfied. For unobfuscatability, we define $\pi(C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}) = \gamma$. Let's verify that γ is pseudorandom given oracle access. As in the proof of Theorem 3.3.11, it follows from Property 3 of Lemma 3.3.6 that a PPT algorithm given oracle access to $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$. will only query $C_{\alpha,(D,\gamma)}$ with negligible probability and hence γ is indistinguishable from uniform.

Finally, let's show that given any circuit C' computing the same function as $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$, we can reconstruct the latter circuit. First, we can decompose $C' = C^1\#D'\#C^2$. Since D' computes the same function as D and $C^1(\alpha) = \beta$, we have $A^{D'}(C^1) = \alpha$, where A is the algorithm from (the modified) Property 2 of Lemma 3.3.6. Given α, we can obtain $\beta = C^1(\alpha)$ and $(D,\gamma) = C^2(\alpha)$, which allows us to reconstruct $C_{\alpha,\beta}\#D\#C_{\alpha,(D,\gamma)}$. □

3.4.2 Approximate obfuscators

One of the most reasonable ways to weaken the definition of obfuscators, is to relax the condition that the obfuscated circuit must compute *exactly* the same function as the original circuit. Rather, we can allow the obfuscated circuit to only *approximate* the original circuit.

We must be careful in defining "approximation". We do not want to lose the notion of an obfuscator as a *general purpose* scrambling algorithm and therefore we want a definition of approximation that will be strong enough to guarantee that the obfuscated circuit can still be used in the place of the original circuit in *any application.* Consider the case of a signature verification algorithm V_K. A polynomial-time algorithm cannot find an input on which V_K does not output 0 (without knowing the signature key). However, we clearly do not want this to mean that the constant zero function is an approximation of V_K.

Definition and Impossibility Result

In order to avoid the above pitfalls we choose a definition of approximation that allows the obfuscated circuit to deviate on a particular input from the original circuit only with negligible probability and allows this event to depend on only the coin tosses of the obfuscating algorithm (rather than over the choice of a randomly chosen input).

Definition 3.4.3. For any function $f : \{0,1\}^n \to \{0,1\}^k$, $\epsilon > 0$, the random variable C is called an *ϵ-approximate implementation* of f if the following holds:

1. C ranges over circuits from $\{0,1\}^n$ to $\{0,1\}^k$
2. For any $x \in \{0,1\}^n$, $\Pr_C[C(x) = f(x)] \geq 1 - \epsilon$

Remark 3.4.4. As mentioned above, obfuscators that relax the functionality requirement to agree with the input circuit on a uniformly distributed *input*, will not suffice for all possible applications. Nonetheless, they may suffice for *some* applications, and so the existence of such obfuscators is a very interesting question (which this work does not answer).

We then define a strongly unobfuscatable function ensemble to be an unobfuscatable function ensemble where the hard property $\pi(f)$ can be computed not only from any circuit that computes f but also from any approximate implementation of f.

Definition 3.4.5. A *strongly unobfuscatable function ensemble* $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ is defined in the same way as an unobfuscatable function ensemble, except that Part 2 of the "unobfuscatability" condition is replaced with the following:

2. $\pi(f)$ is easy to compute with access to a circuit that approximates f: There exists a PPT A and a polynomial $p(\cdot)$ such that for any $f \in \bigcup_{n\in\mathbb{N}} \mathrm{Supp}(\mathcal{H}_n)$ and for any random variable C that is an ϵ-approximate implementation of f

$$\Pr[A(C) = \pi(f)] \geq 1 - \epsilon \cdot p(n)$$

Our main theorem in this section is the following:

Theorem 3.4.6. *If one-way functions exist, then there exists a strongly unobfuscatable function ensemble.*

Similarly to the way that Theorem 3.3.11 implies Theorem 3.3.8, Theorem 3.4.6 implies that, assuming the existence of one-way functions, an even weaker definition of circuit obfuscators (one that allows the obfuscated circuit to only approximate the original circuit) is impossible to meet. We note that it some (but not all) applications of obfuscators, a weaker notion of approximation might suffice. Specifically, in some cases it suffices for the obfuscator to only approximately preserve functionality with respect to a particular distribution on inputs, such as the uniform distribution. (This is implied, but apparently weaker, than the requirement of Definition 3.4.3 — if C is an ε-approximate implementation of f, then for for any fixed distribution D on inputs, C and f agree on a $1-\sqrt{\varepsilon}$ fraction of D with probability at least $1-\sqrt{\varepsilon}$.) We do not know whether approximate obfuscators with respect to this weaker notion exist, and leave it as an open problem.

We shall prove this theorem in the following stages. First we will see why the proof of Theorem 3.3.11 does not apply directly to the case of approximate implementations. Then we shall define a construct called *invoker-randomizable pseudorandom functions*, which will help us modify the original proof to hold in this case.

Generalizing the Proof of Theorem 3.3.11 to the Approximate Case

The first question is whether the proof of Theorem 3.3.11 already shows that the ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ defined there is actually a *strongly* unobfuscatable function ensemble. As we explain below, the answer is no.

To see why, let us recall the definition of the ensemble $\{\mathcal{H}_k\}_{k\in\mathbb{N}}$ that is defined there and uses the distributions $\mathcal{F}_k$ and $\mathcal{G}_k$ that are defined in the proof of Theorem 3.3.8. The distribution $\mathcal{H}_k$ is defined by taking an element from $\mathcal{F}_k$ or $\mathcal{G}_k$, with probability 1/2 each. The distribution $\mathcal{F}_k$ is defined by choosing $\alpha, \beta \leftarrow_{\mathrm{R}} \{0,1\}^k$, a function $D \leftarrow_{\mathrm{R}} \mathcal{D}_{\alpha,\beta}$ and outputting $C_{\alpha,\beta}\#D$. Similarly, $\mathcal{G}_k$ is defined by choosing $\alpha, \beta \leftarrow_{\mathrm{R}} \{0,1\}^k$, $D \leftarrow_{\mathrm{R}} \mathcal{D}_{\alpha,\beta}$ and outputting $Z_k\#D$. The property π is defined simply to distinguish functions in $\mathcal{F}_k$ from those in $\mathcal{G}_k$.

That proof gave an algorithm A' which computes $\pi(f)$ given a circuit computing any function f from $\mathcal{H}$. Let us see why A' might fail when given only an approximate implementation of f. On input a circuit F, A' works as follows: It decomposes F into two circuits $F = F_1\#F_2$. F_2 is used only in a black-box manner, but the queries A' makes to it depend on the gate structure of the circuit F_1. The problem is that a vicious approximate implementation for a function $C_{\alpha,\beta}\#D \in \mathrm{Supp}(\mathcal{F}_k)$ may work in the following way: choose a random circuit F_1 out of some set $\mathcal{C}$ of exponentially many circuits that compute $C_{\alpha,\beta}$, and take F_2 that computes D. Then see at which points A' queries F_2 when given $F_1\#F_2$ as input.[8] As these places depend on F_1, it is possible that for each $F_1 \in \mathcal{C}$, there is a point $x(F_1)$ such that A' will query F_2 at the point $x(F_1)$, but $x(F_1) \neq x(F_1')$ for any

[8] Recall that A' is not some given algorithm that we must treat as a black-box but rather a specific algorithm that we defined ourselves.

$F_1' \in \mathcal{C} \setminus \{F_1\}$. If the approximate implementation changes the value of F_2 at $x(F_1)$, then A''s computation on $F_1\#F_2$ is corrupted.

One way to solve this problem would be to make the queries that A' makes to F_2 *independent* of the structure of F_1. If A' had this property, then given an ϵ-approximate implementation of $C_{\alpha,\beta}\#D$, each query of A' would have only an ϵ chance to get an incorrect answer and overall A' would succeed with probability $1-\epsilon \cdot p(k)$ for some polynomial $p(\cdot)$. (Note that the probability that $F_1(\alpha)$ changes is at most ϵ.)

We will not be able to achieve this, but something slightly weaker that still suffices. Let's look more closely at the structure of $\mathcal{D}_{\alpha,\beta}$ which is defined in the proof of Lemma 3.3.6. We defined there the algorithm

$$D_{K,\alpha,\beta} \stackrel{def}{=} E_{K,\alpha}\#\mathrm{Hom}_K\#B_{K,\beta}$$

and turned it into a deterministic function by using a pseudorandom function f'_K and defining $D'_{K,\alpha,\beta,K'}$ to be the deterministic algorithm that on input $x \in \{0,1\}^q$ evaluates $D_{K,\alpha,\beta}(x)$ using randomness $f_{K'}(x)$. We then defined $\mathcal{D}_{\alpha,\beta}$ to be $D'_{K,\alpha,\beta,K'} = E'_{K,\alpha,K'}\#\mathrm{Hom}'_{K,K'}\#B_{K,\beta}$ for uniformly selected private key K and seed K'.

Now our algorithm A' (that uses the algorithm A defined in Lemma 3.3.6) treats F_2 as three oracles: E, H, and B , where if F_2 computes $D = E'_{K,\alpha,K'}\#\mathrm{Hom}'_{K,K'}\#B_{K,\beta}$ then E is the oracle to $E'_{K,\alpha,K'}$, H is the oracle to $\mathrm{Hom}'_{K,K'}$ and B is the oracle to $B_{K,\beta}$. The queries to E are at the places $1,\ldots,k$ and so are independent of the structure of F_1. The queries that A makes to the H oracle, however, do depend on the structure of F_1.

Recall that any query A' makes to the H oracle are of the form $(c,d,\odot)$ where c and d are ciphertexts of some bits, and $\odot$ is a 4-bit description of a binary boolean function. Just for motivation, suppose that A' has the following ability: given an encryption c, A' can generate a random encryption of the same bit (i.e., distributed according to $\mathrm{Enc}_K(\mathrm{Dec}_K(c), r)$ for uniformly selected r). For instance, this would be true if the encryption scheme were "random self-reducible." Suppose now that, before querying the H oracle with $(c,d,\odot)$, A' generates c',d' that are random encryptions of the same bits as c,d and query the oracle with $(c',d',\odot)$ instead. We claim that if F_2 is an ϵ-approximate implementation of D, then for any such query, there is at most a 64ϵ probability for the answer to be wrong *even if* $(c,d,\odot)$ *depend on the circuit* F. The reason is that the distribution of the modified query $(c',d',\odot)$ depends only on $(\mathrm{Dec}_K(c),\mathrm{Dec}_K(d),\odot)$, and there are only $2\cdot 2\cdot 2^4 = 64$ possibilities for the latter. For each of the 64 possibilities, the probability of an incorrect answer (over the choice of F) is at most ϵ. Choosing $(\mathrm{Dec}_K(c),\mathrm{Dec}_K(d),\odot)$ after F to maximize the probability of an incorrect answer multiplies this probability by at most 64.

We shall now use this motivation to fix the function D so that A' will essentially have this desired ability of randomly self-reducing any encryption to a random encryption of the same bit. Recall that $\mathrm{Hom}'_{K,K'}(c,d,\odot) = \mathrm{Enc}_K(\mathrm{Dec}_K(c)\odot\mathrm{Dec}_K(d); f_{K'}(c,d,\odot))$. Now, a naive approach to ensure that any query returns a random encryption of $\mathrm{Dec}_K(c)\odot\mathrm{Dec}_K(d)$ would be to change the definition of Hom' to the following: $\mathrm{Hom}'_{K,K'}(c,d,\odot,r) = \mathrm{Enc}_K(\mathrm{Dec}_K(c)\odot\mathrm{Dec}_K(d); r)$. Then we change A' to an algorithm A'' that chooses a uniform $r \in \{0,1\}^n$ and thereby ensures that the result is a random encryption of $\mathrm{Dec}_K(c)\odot\mathrm{Dec}_K(d)$. The problem is that this construction would no longer satisfy Property 3 of Lemma 3.3.6 (security against a simulator with oracle access). This is because the simulator could now control the random coins of the encryption scheme and use this to break it. Our solution will be to redefine Hom' in the following way:

$$\mathrm{Hom}'_{K,K'}(c,d,\odot,r) = \mathrm{Enc}_K(\mathrm{Dec}_K(c)\odot\mathrm{Dec}_K(d); f_{K'}(c,d,\odot,r))$$

but require an additional special property from the pseudorandom function $f_{K'}$.

Invoker-Randomizable Pseudorandom Functions

The property we would like the pseudorandom function $f_{K'}$ to possess is the following:

Definition 3.4.7. A function ensemble $\{f_{K'}\}_{K' \in \{0,1\}^*}$ ($f_{K'} : \{0,1\}^{q+n} \rightarrow \{0,1\}^n$, n ,q polynomially related to $|K'|$) is called an *invoker-randomizable pseudorandom function ensemble* if the following holds:

1. $\{f_{K'}\}_{K' \in \{0,1\}^*}$ is a pseudorandom function ensemble
2. For any $x \in \{0,1\}^q$, if r is chosen uniformly in $\{0,1\}^n$ then $f_{K'}(x,r)$ is distributed uniformly (and so independently of x) in $\{0,1\}^n$.

Fortunately, we can prove the following lemma:

Lemma 3.4.8. *If pseudorandom functions exist then there exist invoker-randomizable pseudorandom functions.*

Proof Sketch: Suppose that $\{g_{K'}\}_{K' \in \{0,1\}^*}$ is a pseudorandom function ensemble and that $\{p_S\}_{S \in \{0,1\}^*}$ is a pseudorandom function ensemble in which for any $S \in \{0,1\}^*$, p_S is a permutation (the existence of such ensembles is implied by the existence of ordinary pseudorandom function ensembles [LR86]).

We define the function ensemble $\{f_{K'}\}_{K' \in \{0,1\}^*}$ in the following way:

$$f_{K'}(x,r) \stackrel{def}{=} p_{g_{K'}(x)}(r)$$

It is clear that this ensemble satisfies Property 2 of Definition 3.4.7 as for any x, the function $r \mapsto f_{K'}(x,r)$ is a permutation.

What needs to be shown is that it is a pseudorandom function ensemble. We do this by showing that for any PPT D, the following probabilities are identical up to a negligible factor.

1. $\Pr_{K'}[D^{f_{K'}}(1^k) = 1]$ (where $k = |K'|$).
2. $\Pr_G[D^{(x,R) \mapsto p_{G(x)}(R)}(1^k) = 1]$, where G is a true random function.
3. $\Pr_{P_1,\ldots,P_t}[D^{P_1,\ldots,P_t}(1^k) = 1]$, where $t = t(k)$ is a bound on the number of queries that D makes and each time D makes a query with a new value of x we use a new random function P_i. (This requires a hybrid argument).
4. $\Pr_F[D^F(1^k) = 1]$, where F is a truly random function.

$\square$

Finishing the Proof of Theorem 3.4.6

Now, suppose we use a pseudorandom function $f_{K'}$ that is invoker-randomizable, and modify the algorithm A' so that all its queries $(c, d, \odot)$ to the H oracle are augmented to be of the form $(c, d, \odot, r)$, where r is chosen uniformly and independently for each query. Then the result of each such query is a *random* encryption of $\mathrm{Dec}_K(c) \odot \mathrm{Dec}_K(d)$. Therefore, as argued above, A' never gets a wrong answer from the H oracle with probability at least $1 - p(k) \cdot \epsilon$, for some polynomial $p(\cdot)$. Indeed, this holds because aside from the first queries which are fixed and therefore independent of the gate structure of F_1, all other queries are of the form $(c, d, \odot, r)$ where c and d are uniformly distributed and independent encryptions of some bits a and b, and r is uniformly distributed. Only $(a, b, \odot)$ depend on the gate structure of F_1, and there are only 64 possibilities for them. Assuming A' never gets an incorrect answer from the H oracle, its last query to the B oracle will be a uniformly distributed encryption of $\beta_1, \ldots, \beta_k$, which is independent of the structure of F_1, and so has only an ϵ probability to be incorrect. This completes the proof.

One point to note is that we have converted our deterministic algorithm A' of Theorem 3.3.11 into a *probabilistic* algorithm.

3.4.3 Impossibility of the applications

So far, we have only proved impossibility of some natural and arguably minimalistic definitions for obfuscation. Yet it might seem that there's still hope for a different definition of obfuscation, one that will not be impossible to meet but would still be useful for some intended applications. We'll show now that this is not the case for many of the applications we described in the introduction. Rather, any definition of obfuscator that would be strong enough to provide them, will be impossible to meet.

Note that we do not prove that the applications themselves are impossible to meet. Indeed, it is widely believed that public key cryptosystems *do* exist. Rather, we prove that there does not exist an obfuscator[9] that can be used to achieve these applications in the ways that are described in Section 3.1.1. Our results in the section also extend to approximate obfuscators.

Consider, for example, the application to transforming private-key encryption to public-key ones. The circuit $\widetilde{E_k}$ in the following definition can be viewed as an encryption-key in the corresponding public-key encryption scheme.

Definition 3.4.9. A private-key encryption scheme (G, E, D) is called *unobfuscatable* if there exists a PPT A such that

$$\Pr_{K \leftarrow_{\mathrm{R}} G(1^k)}[A(\widetilde{E_K}) = K] \geq 1 - \mathsf{neg}(k)$$

where $\widetilde{E_K}$ is any circuit that computes the encryption function with private key K.

Note that an unobfuscatable encryption scheme is unobfuscatable in a very strong sense. An adversary is able to completely break the system given *any* circuit that computes the encryption algorithm.

We prove in Theorem 3.4.13 that if encryption schemes exist, then so do unobfuscatable encryption schemes that satisfy the same security requirements.[10] This means that any definition of an obfuscators that will be strong enough to allow the conversion of private-key encryption schemes into public-key encryption schemes mentioned in Section 3.1.1, would be impossible to meet (because there exist unobfuscatable encryption schemes).

We present analogous definitions for unobfuscatable signature schemes, MACs, and pseudorandom functions.

Definition 3.4.10. A signature scheme (G, S, V) is called *unobfuscatable* if there exists a PPT A such that

$$\Pr_{(SK,VK) \leftarrow_{\mathrm{R}} G(1^k)}[A(\widetilde{S_{SK}}) = SK] \geq 1 - \mathsf{neg}(k)$$

[9] By this, we mean any algorithm that satisfies the syntactic requirements of Definition 3.2.2 (functionality and polynomial slowdown).

[10] Recall that, for simplicity, we only consider deterministic encryption schemes here and relaxed notions of security that are consistent with them (cf., Footnote 3).

where $\widetilde{S_{SK}}$ is any circuit which computes the signature function with signing key SK.

Definition 3.4.11. A message authentication scheme (G, S, V) is called *unobfuscatable* if there exists a PPT A such that

$$\Pr_{K \leftarrow_{\mathrm{R}} G(1^k)}[A(\widetilde{S_K}) = K] \geq 1 - \mathsf{neg}(k)$$

where $\widetilde{S_K}$ is any circuit which computes the tagging function with tagging key K.

Definition 3.4.12. A pseudorandom function ensemble $\{h_K\}_{K \in \{0,1\}^*}$ is called *unobfuscatable* if there exists a p.p.t A such that

$$\Pr_{K \leftarrow_{\mathrm{R}} \{0,1\}^k}[A(\widetilde{H_K}) = K] \geq 1 - \mathsf{neg}(k)$$

where $\widetilde{H_K}$ is any circuit that computes h_K.

One implication of the existence of unobfuscatable pseudorandom function ensembles is that for many *natural* protocols that are secure in the random oracle model (such as the Fiat–Shamir authentication protocol [FS86]), one can find a pseudorandom function ensemble $\{h_k\}_{k \in \{0,1\}^*}$ such that if the random oracle is replaced with *any* circuit that computes h_k, the protocol would not be secure.[11]

Theorem 3.4.13. *1. If signature schemes exist, then so do unobfuscatable signature schemes.*

2. If private-key encryption schemes exist, then so do unobfuscatable encryption schemes.

3. If pseudorandom function ensembles exist, then so do unobfuscatable pseudorandom function ensembles.

4. If message authentication schemes exist, then so do unobfuscatable message authentication schemes.

Proof Sketch: First note that the existence of any one of these primitives implies the existence of one-way functions [IL89]. Therefore, Theorem 3.4.2 gives us a totally unobfuscatable function ensemble $\mathcal{H} = \{\mathcal{H}_k\}$.

Now, we shall sketch the construction of the unobfuscatable signature scheme. All other constructions are similar. Take an existing signature scheme (G, S, V) (where G is the key generation algorithm, S the signing algorithm, and V the verification algorithm). Define the new scheme (G', S', V') as follows:

The generator G' on input 1^k uses the generator G to generate signing and verifying keys $(SK, VK) \leftarrow_{\mathrm{R}} G(1^k)$. It then samples a circuit $f \leftarrow_{\mathrm{R}} \mathcal{H}_\ell$, where $\ell = |SK|$. The new signing key SK' is (SK, f) while the verification key VK' is the same as VK.

We can now define

$$S'_{SK,f}(m) \stackrel{def}{=} (S_{SK}(m), f(m), SK \oplus \pi(f)),$$

where π is the function from the unobfuscatability condition in Definition 3.4.1.

$$V'_{VK}(m, (\tau, x)) \stackrel{def}{=} V_{VK}(m, \tau)$$

[11] In contrast, the results of Chapter 4 and [GT03] imply the existence of *particular* (arguably "unnatural") protocols for which the Fiat-Shamir heuristic will fail if the random oracle is replaced with *any* efficiently computable function.

We claim that (G', S', V') is an unobfuscatable, yet secure, signature scheme. Clearly, given any circuit that computes $S'_{SK,f}$, one can obtain $SK \oplus \pi(f)$ and a circuit that computes the same function as f. Possession of the latter enables one to reconstruct the original circuit f itself, from which $\pi(f)$ and then SK can be computed.

To see that scheme (G', S', V') retains the security of the scheme (G, S, V), observe that being given oracle access to $S'_{SK,f}$ is equivalent to being given oracle access to S_{SK} and f, along with being given the string $\pi(f) \oplus SK$. Using the facts that $\pi(f)$ is indistinguishable from random given oracle access to f and that f is chosen independently of SK, it can be easily shown that the presence of f and $\pi(f) \oplus SK$ does not help an adversary break the signature scheme.

The construction of an unobfuscatable encryption scheme and pseudorandom function ensemble is similar. The only detail is that when we construct the pseudorandom function ensemble, we need to observe that Theorem 3.4.2 can be modified to give $\mathcal{H}$ which is also a family of pseudorandom functions. (To do this, all places where the functions f in $\mathcal{H}$ were defined to be zero should instead be replaced with values of a pseudorandom function.) □

3.4.4 Obfuscating restricted circuit classes

Given our impossibility results for obfuscating general circuits, one may ask whether it is easier to obfuscate computationally restricted classes of circuits. Here we argue that this is unlikely for all but very weak models of computation.

Theorem 3.4.14. *If factoring Blum integers is "hard"*[12] *then there is a family $\mathcal{H}_k$ of unobfuscatable functions such that every $f \leftarrow_R \mathcal{H}_k$ is computable by a constant-depth threshold circuit of size* $\mathrm{poly}(k)$ *(i.e., in* $\mathbf{TC_0}$*).*

Proof Sketch: Naor and Reingold [NR97] showed that under the stated assumptions, there exists a family of pseudorandom functions computable in $\mathbf{TC_0}$. Thus, we simply need to check that we can build our unobfuscatable functions from such a family without a substantial increase in depth. Recall that the unobfuscatable function ensemble $\mathcal{H}_k$ constructed in the proof of Theorem 3.3.11 consists of functions of the form $C_{\alpha,\beta}\#D$ or $Z_k\#D$, where D is from the family $\mathcal{D}_{\alpha,\beta}$ of Lemma 3.3.6. It is easy to see that $C_{\alpha,\beta}$ and Z_k are in $\mathbf{TC_0}$, so we only need to check that $\mathcal{D}_{\alpha,\beta}$ consists of circuits in $\mathbf{TC_0}$. The computational complexity of circuits in the family $\mathcal{D}_{\alpha,\beta}$ is dominated by performing encryptions and decryptions in a private-key encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ and evaluating a pseudorandom function $f_{K'}$ which is used to derandomize the probabilistic circuit $D_{K,\alpha,\beta}$. If we use the Naor–Reingold pseudorandom functions both for $f_{K'}$ and to construct the encryption scheme (in the usual way, setting $\mathrm{Enc}_K(b) = (r, f_K(r) \oplus b)$), then the resulting circuit is in $\mathbf{TC_0}$. □

3.4.5 Relativization

In this section, we discuss whether our results relativize. To do this, we must clarify the definition of an obfuscator relative to an oracle $F : \{0,1\}^* \to \{0,1\}^*$. What we mean is that all algorithms in the definition, including the one being obfuscated and including the adversary, have oracle access to F. For a circuit, this means that the circuit can have gates for evaluating F. We fix an encoding of (oracle) circuits as binary strings such that a circuit described by a string of length s can only make oracle queries of total length at most s.

[12] This result is also implied if the Decisional Diffie–Hellman problem is "hard"; see [NR97] for precise statements of these assumptions.

By inspection, our initial (easy) impossibility results hold relative to any oracle, as the involve only simulation and diagonalization.

Proposition 3.4.15. *Proposition 3.3.4 (impossibility of 2-circuit obfuscators) and Theorem 3.3.5 (impossibility of TM obfuscators) hold relative to any oracle.*

Interestingly, however, our main impossibility results do not relativize.

Proposition 3.4.16. *There is an oracle relative to which efficient circuit obfuscators exist. Thus, Theorem 3.3.8, Theorem 3.3.11, and Corollary 3.3.10 do not relativize.*

This can be viewed both as evidence that these results are nontrivial, and as (further) evidence that relativization is not a good indication of what we can prove.

Proof Sketch: The oracle $F = \bigcup_k F_k$ will consist of two parts $F_k = \mathcal{O}_k \# \mathrm{E}_k$, where $\mathcal{O}_k : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^{6k}$, and $\mathrm{E}_k : \{0,1\}^{6k} \times \{0,1\}^k \to \{0,1\}^k$. $\mathcal{O}_k$ is simply a uniformly random injective function of the given parameters. $\mathrm{E}_k(x,y)$ is defined as follows: If there exists a (C,r) such that $\mathcal{O}_k(C,r) = x$, then $\mathrm{E}_k(x,y) = C^F(y)$ (where C is viewed as the description of a circuit). Otherwise, $\mathrm{E}_k(x,y) = \perp$. Note that this definition of F_k is not circular, because C can only make oracle queries of size at most $|C| = k$, and hence can only query $F_{k'}$ for $k' \leq k/2$.

Now we can view $x = \mathcal{O}_k(C,r)$ as an obfuscation of C using coin tosses r. This satisfies the syntactic requirements of obfuscation, since $|x| = O(|C|)$ and the E_k allows one to efficiently evaluate $C(y)$ given just x and y. (Technically, we should define the obfuscation of C to be a circuit which has x hardwired in and makes an oracle query to E_k.)

So we only need to prove the virtual black-box property. By a union bound over polynomial-time adversaries A of description size smaller than $k/2$ and circuits C of size k, it suffices to prove the following claim.[13]

Claim 3.4.17. *For every PPT A there exists a PPT S such that for every circuit C of size k, the following holds with probability at least $1 - 2^{-2k}$ over F:*

$$\left| \Pr_{r \leftarrow_R \{0,1\}^k} \left[A^F(\mathcal{O}_k(C,r)) = 1 \right] - \Pr \left[S^{F,C}(1^k) = 1 \right] \right| \leq 2^{-\Omega(k)}$$

Fix a PPT A. We define the simulator S as follows. $S^{F,C}(1^k)$ chooses $x \leftarrow_{\mathrm{R}} \{0,1\}^{6k}$ and simulates $A^F(x)$, using its own F-oracle to answer A's oracle queries, *except* A's queries to $\mathrm{E}_{k'}$ for $k' \geq k$. On A's query (x', y') to $\mathrm{E}_{k'}$, S feeds A the response z computed as follows:

1. If $x' = x$, then set $z = C(y')$ (computed using oracle access to C).
2. Else if $x' = \mathcal{O}_{k'}(C', r')$ for some previous query (C', r') to the $\mathcal{O}_{k'}$-oracle, then set $z = (C')^F(y')$ (computed recursively using these same rules).
3. Else set $z = \perp$.

From the fact that a circuit of size s can only make oracle queries of total length s, it follows that the recursive evaluation of $(C')^F(y)$ only incurs a polynomial overhead in running time. Also note that S never queries the $\mathrm{E}_{k'}$ oracle for $k' \geq k$.

[13]Note that we are only proving the virtual black-box property against adversaries of "bounded nonuniformity," which in particular includes all uniform PPT adversaries. Presumably it can also be proven against nonuniform adversaries, but we stick to uniform adversaries for simplicity.

Let us denote the execution of the above simulation for a particular x by $S^{F,C}(x)$. Notice that when $x = \mathcal{O}_k(C, r)$ for some r, then $S^{F,C}(x)$ and $A^F(x)$ have exactly the same behavior *unless* the above simulation produces some query (x', y') such that $x' \in \text{Image}(\mathcal{O}_{k'})$, $x' \neq x$, and x' was not obtained by a previous query to $\mathcal{O}_{k'}$. Since $\mathcal{O}$ is a random length-tripling function, it follows that the latter happens with probability at most $\text{poly}(k) \cdot 2^{2k}/2^{6k}$, taken over the choice of F and a random r (recall that $x = \mathcal{O}_k(C, r)$).[14] Thus, with probability at least $1 - 2^{-3k}$ over the choice of F, $S^{F,C}(\mathcal{O}_k(C, r)) = A^F(\mathcal{O}_k(C, r))$ for all but a $2^{-\Omega(k)}$ fraction of r's.

Thus, proving Claim 3.4.17 reduces to showing that:

$$\left| \Pr_{r \leftarrow_{\mathrm{R}} \{0,1\}^k} \left[S^{F,C}(\mathcal{O}_k(C, r)) = 1 \right] - \Pr_{x \leftarrow_{\mathrm{R}} \{0,1\}^{6k}} \left[S^{F,C}(x) = 1 \right] \right| \leq 2^{-\Omega(k)}$$

with high probability (say, $1 - 2^{3k}$) over the choice of F.

In other words, we need to show that the function $G(r) \stackrel{def}{=} \mathcal{O}_k(C, r)$ is a pseudorandom generator against S. Since G is a random function from $\{0,1\}^k \to \{0,1\}^{6k}$, this would be obvious were it not for the fact that S has oracle access to F (which is correlated with G). Recall, however, that we made sure that S does not query the $\mathrm{E}_{k'}$-oracle for any $k' \geq k$. This enables us to use the following lemma (proven below):

Lemma 3.4.18. *There is a constant $\delta > 0$ such that the following holds for all sufficiently large K and any $L \geq K^2$. Let D be an algorithm that makes at most K^δ oracle queries and let G be a random injective function $G : [K] \to [L]$. Then with probability at least $1 - 2^{-K^\delta}$ over G,*

$$\left| \Pr_{x \in [K]} \left[D^G(G(x)) = 1 \right] - \Pr_{y \in [L]} \left[D^G(y) = 1 \right] \right| \leq \frac{1}{K^\delta} \tag{3.4}$$

Let us see how Lemma 3.4.18 implies what we want. Let $K = 2^k$ and associate $[K]$ with $\{0,1\}^k$. We fix all values of $\mathcal{O}_{k'}$ for all $k' \neq k$ and $\mathrm{E}_{k'}$ for all $k' < k$. We also fix the values of $\mathcal{O}_k(C', r)$ for all $C' \neq C$, and view $G(r) \stackrel{def}{=} \mathcal{O}_k(C, r)$ as a random injective function from $[K]$ to the remaining $L = K^6 - (K-1) \cdot K$ elements of $\{0,1\}^{6k}$. The only oracle queries of S that vary with the choice of G are queries to $\mathcal{O}_k$ at points of the form (C, r), which is equivalent to queries to G. Thus, Lemma 3.4.18 implies that the output of G is indistinguishable from the uniform distribution on some subset of $\{0,1\}^{6k}$ of size L. Since the latter has statistical difference $(K^6 - L)/K^6 < 1/K^4$ from the uniform distribution on $\{0,1\}^{6k}$, we conclude that G is ε-pseudorandom (for $\varepsilon = 1/K^\delta + 1/K^4 = 2^{-\Omega(k)}$) against S with probability at least $1 - 2^{-K^\delta} > 1 - 2^{-3k}$, as desired. $\square$

Proof of Lemma 3.4.18

We prove the lemma via a counting argument in the style of Gennaro and Trevisan's proof that a random permutation is one-way against nonuniform adversaries [GT00]. Specifically, we will show that "most" G for which Inequality (3.4) fails have a "short" description given D, and hence there cannot be too many of them.

Let $\mathcal{G}$ be the collection of G's for which Inequality (3.4) fails (for a sufficiently small δ, whose value is implicit in the proof below). We begin by arguing that, for every $G \in \mathcal{G}$, there is a large set $S_G \subset [K]$ of inputs on which D's behavior is "independent," in the sense that for $x \in S_G$, none of the oracle queries made in the execution of $D^G(G(x))$ are at points in S_G, yet D still has nonnegligible advantage in distinguishing $G(x)$ from random. Actually, we will not be able to afford

[14]Technically, this probability (and later ones in the proof) should also be taken over the coin tosses of A/S.

specifying S_G when we "describe" G, so we actually show that there is a fixed set S (independent of G) such that for most G, the desired set S_G can be obtained by just throwing out a small number of elements from S.

Claim 3.4.19. *There is a set $S \subset [K]$ with $|S| = K^{1-5\delta}$, and $\mathcal{G}' \subset \mathcal{G}$ with $|\mathcal{G}'| = |\mathcal{G}|/2$ such that for all $G \in \mathcal{G}'$, there is a set $S_G \subset S$ with the following properties:*

1. *$|S_G| = (1-\gamma)|S|$, where $\gamma = K^{-3\delta}$.*

2. *If $x \in S_G$, then $D^G(G(x))$ never queries its oracle at an element of S_G.*

3. *$\left|\Pr_{x\in S_G}\left[D^G(G(x)) = 1\right] - \Pr_{y\in L_G}\left[D^G(y) = 1\right]\right| > \frac{1}{2K^\delta}$, where $L_G \stackrel{def}{=} [L]\backslash G([K]\backslash S_G)$. (Note that L_G contains more than a $1 - K/L$ fraction of L.)*

Proof. First consider choosing both a random $G \leftarrow_{\mathrm{R}} \mathcal{G}$ and a random S (among subsets of $[K]$ of size $K^{1-5\delta}$). We will show that with probability at least 1/2, there is a good subset $S_G \subset S$ satisfying Properties 1–3. By averaging, this implies that there is a fixed set S for which a good subset exists for at least half the $G \in \mathcal{G}$, as desired. Let's begin with Property 2. For a random G, S, and a random $x \in S$, note that $D^G(G(x))$ initially has no information about S, which is a random set of density $K^{-5\delta}$. Since D makes at most K^δ queries, the probability that it queries its oracle at some element of S is at most $K^\delta \cdot K^{-5\delta} = K^{-4\delta}$. Thus, with probability at least 3/4 over G and S, $D^G(G(x))$ queries its oracle at an element of S for at most a $4/K^{-4\delta} < \gamma$ fraction of $x \in S$. Throwing out this γ fraction of elements of S gives a set S_G satisfying Properties 1 and 2.

Now let's turn to Property 3. By a Chernoff-like bound, with probability at least $1-\exp(\Omega(K^{1-5\delta}\cdot (K^{-\delta})^2)) > 3/4$ over the choice of S,

$$\left|\Pr_{x\in S}\left[D^G(G(x)) = 1\right] - \Pr_{x\in[K]}\left[D^G(G(x)) = 1\right]\right| \le \frac{1}{4K^\delta}.$$

Then we have:

$$\begin{aligned}
&\left|\Pr_{x\in S_G}\left[D^G(G(x)) = 1\right] - \Pr_{y\in L_G}\left[D^G(y) = 1\right]\right| \\
\ge\ & \left|\Pr_{x\in[K]}\left[D^G(G(x)) = 1\right] - \Pr_{y\in[L]}\left[D^G(y) = 1\right]\right| \\
&- \left|\Pr_{x\in S_G}\left[D^G(G(x)) = 1\right] - \Pr_{x\in[S]}\left[D^G(G(x)) = 1\right]\right| \\
&- \left|\Pr_{x\in S}\left[D^G(G(x)) = 1\right] - \Pr_{x\in[K]}\left[D^G(G(x)) = 1\right]\right| \\
&- \left|\Pr_{y\in[L]}\left[D^G(y) = 1\right] - \Pr_{y\in L_G}\left[D^G(y) = 1\right]\right| \\
>\ & 1/K^\delta - \gamma - 1/4K^\delta - K/L \\
>\ & 1/2K^\delta
\end{aligned}$$

□

Now we show how the above claim implies that every $G \in \mathcal{G}'$ has a "small" description.

Claim 3.4.20. *Every $G \in \mathcal{G}'$ can be uniquely described by $(\log B) - \Omega(K^{1-7\delta})$ bits given D, where B is the number of injective functions from $[K]$ to $[L]$.*

Proof. For starters, the description of G will contains the set S_G and the values of $G(x)$ for all $x \notin S_G$. Now we'd like to argue that this information is enough to determine $D^G(y)$ for all y. This won't exactly be the case, but rather we'll show how to compute $M^G(y)$ for some M that is "as good" as D. From Property 3 in Claim 3.4.19, we have

$$\Pr_{x \in S_G}\left[D^G(G(x)) = 1\right] - \Pr_{y \in L_G}\left[D^G(y) = 1\right] > \frac{1}{2K^\delta}.$$

(We've dropped the absolute values. The other case is handled analogously, and the only cost is one bit to describe which case holds.) We will describe an algorithm M for which the same inequality holds, yet M will only use the information in our description of G instead of making oracle queries to G. Specifically, on input y, M simulates $D(y)$, except that it handles each oracle query z as follows:

1. If $z \notin S_G$, then M responds with $G(z)$ (This information is included in our description of G).
2. If $z \in S_G$, then M halts and outputs 0. (By Property 2 of Claim 3.4.19, this cannot happen if $y \in G(S_G)$, hence outputting 0 only improves M's distinguishing gap.)

Thus, given S_G and $G|_{[K]\setminus S_G}$, we have a function M satisfying

$$\Pr_{x \in S_G}[M(G(x)) = 1] - \Pr_{y \in L_G}[M(y) = 1] > \frac{1}{2K^\delta} \tag{3.5}$$

To complete the description of G, we must specify $G|_{S_G}$, which we can think of as first specifying the image $T = G(S_G) \subset L_G$ and then the bijection $G : S_G \to T$. However, we can save in our description because T is constrained by Inequality (3.5), which can be rewritten as:

$$\Pr_{y \in T}[M(y) = 1] - \Pr_{y \in L_G}[M(y) = 1] > \frac{1}{2K^\delta} \tag{3.6}$$

Chernoff Bounds say that most large subsets are good approximators of the average of a boolean function. Specifically, at most a $\exp(-\Omega((1-\gamma)K^{1-5\delta} \cdot (K^{-\delta})^2)) = \exp(-\Omega(K^{1-7\delta}))$ fraction of sets $T \subset L_G$ of size $(1-\gamma)K^{1-5\delta}$ satisfy Equation 3.6.

Thus, using M, we have "saved" $\Omega(K^{1-7\delta})$ bits in describing $G(S_G)$ (over the standard "truth-table" representation of a function G). However, we had to describe the set S_G itself, which would have been unnecessary in the truth-table representation. Fortunately, we only need to describe S_G as a subset of S, and this only costs $\log\binom{K^{1-5\delta}}{(1-\gamma)K^{1-5\delta}} = O(H_2(\gamma)K^{1-5\delta}) < O(K^{1-8\delta}\log K)$ bits (where $H_2(\gamma) = O(\gamma \log(1/\gamma))$ denotes the binary entropy function). So we have a net savings of $\Omega(K^{1-7\delta}) - O(K^{1-8\delta}\log K) = \Omega(K^{1-7\delta})$ bits. □

From Claim 3.4.20, $\mathcal{G}'$ can consist of at most an $\exp(-\Omega(K^{1-7\delta})) < K^{-\delta}/2$ fraction of injective functions $[K] \to [L]$, and thus $\mathcal{G}$ has density smaller than $K^{-\delta}$, as desired. □

Bounded relativization.

While our result does not relativize in the usual sense, the proof does work for a slightly different form of relativization, which we refer to as *bounded relativization* (and is how the Random Oracle Model is sometimes interpreted in cryptography.) In *bounded relativization*, an oracle is a *finite* function with fixed input length (polynomially related to the security parameter k), and all algorithms/circuits in the protocol can have running time larger than this length (but still polynomial in k). In particular, in the context of obfuscation, this means that the circuit to be obfuscated can have size polynomial in this length.

Proposition 3.4.21. *Theorems 3.3.11 and 3.3.8 (one-way functions imply unobfuscatable functions and impossibility of circuit obfuscators), and Corollary 3.3.10 (unconditional impossibility of efficient circuit obfuscators) hold under bounded relativization (for any oracle).*

Proof Sketch: The only modification needed in the construction is to deal with oracle gates in the Hom algorithm in the proof of Lemma 3.3.6. Let's call say the oracle F has input length ℓ and output length 1 (without loss of generality). We augment the Hom_K to also take inputs of the form $(c_1, \ldots, c_\ell, \texttt{oracle})$ (where $(c_1, \ldots, c_\ell)$ are ciphertexts), on which it naturally outputs $\mathrm{Enc}_K(F(\mathrm{Dec}_K(c_1), \mathrm{Dec}_K(c_2), \ldots, \mathrm{Dec}_K(c_\ell)))$. The rest of the proof proceeds essentially unchanged. $\square$

Chapter 4

Non-Black-Box Zero Knowledge

Summary: The *simulation paradigm* is central to cryptography. A *simulator* is an algorithm that tries to simulate the interaction of the adversary with an honest party, without knowing the private input of this honest party. Almost all known simulators use the adversary's algorithm as a *black-box*. We present the first constructions of non-black-box simulators. Using these new non-black-box techniques we obtain several results that were previously shown to be impossible to obtain using black-box simulators.

Specifically, assuming the existence of collision-resistent hash functions, we construct a new zero-knowledge argument system for **NP** that satisfies the following properties:

1. This system has a constant number of rounds with negligible soundness error.
2. It remains zero knowledge even when composed concurrently n times, where n is the security parameter.

   Simultaneously obtaining Properties 1 and 2 has been proven to be impossible to achieve using black-box simulators.
3. It is an Arthur-Merlin (public coins) protocol.

   Simultaneously obtaining Properties 1 and 3 has also been proven to be impossible to achieve with a black-box simulator.
4. It has a simulator that runs in *strict* polynomial time, rather than in *expected* polynomial time.

   All previously known zero-knowledge arguments satisfying Property 1 utilized *expected* polynomial-time simulators. In Chapter 5, we show that simultaneously obtaining Properties 1 and 4 is also impossible to achieve with a black-box simulator.

4.1 Introduction

The *simulation paradigm* is one of the most important paradigms in the definition and design of cryptographic primitives. For example, this paradigm arises in a setting in which two parties, Alice and Bob, interact and Bob knows a secret. We want to make sure that Alice hasn't learned anything about Bob's secret as the result of this interaction, and do so by showing that Alice could

This chapter is based on the paper [Bar01]. Section 4.5 is based on the paper [BG01], which is joint work with Oded Goldreich.

have *simulated the entire interaction by herself.* Therefore, she has gained no further knowledge as the result of interacting with Bob, beyond what she could have discovered by herself.

The canonical example of the simulation paradigm is its use in the definition of *zero-knowledge proofs*, as presented by Goldwasser, Micali and Rackoff [GMR85]. Suppose that both Alice and Bob know a public graph G, and in addition Bob knows a Hamiltonian cycle C in this graph. In a zero-knowledge proof, Bob manages to *prove* to Alice that the graph G contains a Hamiltonian cycle, and yet Alice has learned nothing about the cycle C, as she could have simulated the entire interaction by herself.

A crucial point is that we do not want Alice to gain knowledge even if she *deviates arbitrarily* from the protocol when interacting with Bob. This is usually formalized in the following way: for every algorithm V^* that represents the strategy of the verifier (Alice), there exists a simulator M^* that can simulate the entire interaction of the verifier and the honest prover (Bob) without access to the prover's auxiliary information (i.e., the Hamiltonian cycle). That is, the simulator only has access to the public information (i.e., the graph) that was known to the verifier (Alice) before she interacted with the prover (Bob).

Consider the simulator's task even in the easier case in which Alice does follow her prescribed strategy. One problem that the simulator faces is that, in general, it is impossible for it to generate a convincing proof that the graph G is hamiltonian, without knowing a Hamiltonian cycle in the graph. How then can the simulator generate an interaction that is indistinguishable from the actual interaction with the prover? The answer is that the simulator has two advantages over the prover, which compensate for the serious disadvantage of (the simulator's) not knowing a Hamiltonian cycle in the graph. The first advantage is that, unlike in the true interaction, the simulator has access to the verifier's random-tape. This means that it can actually determine the next question that the verifier is going to ask. The second advantage is that, unlike in the actual interaction, the simulator has many attempts at answering the verifier's questions. This is because if it fails, it can simply choose not to output this interaction but rather retry again and output only the "take" in which it succeeds. This is in contrast to an actual proof, where if the party attempting to prove failed even once to answer a question then the proof would be rejected. The difference is similar to the difference between a live television show and a taped show. For example, if someone has a 10% probability of success in shooting a basketball, then he will probably never have 10 straight hits in his life. In contrast, using video-editing, it is very easy to create a film where this person has 10 straight hits. This second technique is called *rewinding* because the simulator that fails to answer a question posed by the verifier, simply rewinds the verifier back and tries again.

All previously known zero-knowledge protocols made use of this rewinding technique in their simulators. However this technique, despite all its usefulness, has some problems. These problems arise mainly in the context of parallel and concurrent compositions. For example, using this technique it is impossible to show that a constant-round zero-knowledge proof[1] remains zero-knowledge under concurrent composition [CKPR01]. It is also impossible to construct a constant-round zero-knowledge proof with a simulator that runs in *strict* polynomial time (rather than *expected* polynomial running time) or a constant-round proof of knowledge with a strict polynomial time knowledge extractor (see Chapter 5).

The reason that all the known simulators were "confined" to the rewinding technique is that it is very hard to take advantage of the knowledge of the verifier's random-tape when using the verifier's strategy as a *black-box.* Let us expand a little on what we mean by this notion. As noted above, to show that a protocol is zero knowledge, one must show that a simulator exists for *every* arbitrary

[1]Here and throughout this paper, we only consider zero-knowledge proofs or arguments that have negligible soundness error.

algorithm V^* that represents the verifier's strategy. Almost all the known protocols simply used a *single generic* simulator that used the algorithm V^* as an oracle (i.e. as a *black-box* subroutine). Indeed it seemed very hard to do anything else, as using V^* in any other way seemed to entail some sort of "reverse-engineering" that is considered a very hard (if not impossible) thing to do.

It can be shown that for black-box simulators, the knowledge of the verifier's random-tape does not help the simulator, because a verifier can have its randomness "hardwired" into its algorithm (for instance in the form of a description of a hash/pseudorandom function). Therefore, black-box simulators are essentially restricted to using the rewinding technique, and so suffer from its consequences. Indeed, as mentioned above, several negative results have been proved about the power of black-box simulators, starting with the results of Goldreich and Krawczyk [GK90] regarding non-existence of *black-box* 3-round zero-knowledge proofs and constant-round Arthur-Merlin zero-knowledge proofs, to the recent result of Canetti, Kilian, Petrank and Rosen [CKPR01] regarding impossibility of black-box constant-round concurrent zero-knowledge.

4.1.1 Our Results

We show that the belief that one can not construct non-black-box simulators is false. That is, given the code of a (possibly cheating) efficient verifier as an auxiliary input, the simulator may significantly use this code in other ways than merely running it, and so obtain goals that are provably impossible to obtain when using the verifier only as a black-box. Specifically, assuming the existence of collision-resistent hash functions, we construct a new zero-knowledge argument (i.e., a computationally-sound proof) for any language in **NP** that satisfies the following properties:

1. It is zero-knowledge with respect to non-uniform adversaries with auxiliary information.

2. It has a constant number of rounds and negligible soundness error.

3. It remains zero-knowledge even if executed concurrently n times, where n is the security parameter. We call a protocol that satisfies this property a *bounded concurrent zero-knowledge* protocol.[2]

4. It is an Arthur-Merlin (public coins) protocol.

5. It has a simulator that runs in *strict* probabilistic polynomial-time, rather than *expected* probabilistic polynomial-time.

The above protocol should be contrasted with the following impossibility results regarding *black-box* zero-knowledge arguments for non-trivial languages: Goldreich and Krawczyk [GK90] showed that such protocols cannot satisfy both Properties 2 and 4. Canetti, Kilian, Petrank and Rosen [CKPR01] showed that such protocols cannot satisfy both Properties 2 and 3. In Chapter 5, we show that such protocols cannot satisfy Properties 2 and 5.

4.1.2 Our approach

Our zero-knowledge argument is constructed using a technique, which we call the FLS technique, that has been used before in the design of zero-knowledge arguments (its first explicit use was by

[2]The choice of n repetitions is quite arbitrary and could be replaced by any *fixed* polynomial (e.g. n^3) in the security parameter. This is in contrast to a standard concurrent zero-knowledge protocol [DNS98, RK99] that remains zero-knowledge when executed concurrently any polynomial number of times.

Feige, Lapidot and Shamir [FLS99]). In the FLS technique, we take an interactive proof/argument for a language L and modify it so that if the prover knows some *trapdoor information* σ, then the prover will be able to "cheat" and convince the verifier that *any* string x is in L, even without knowing a witness for x and even when x is actually not in L. Naturally, to preserve soundness, one must ensure that it is infeasible to obtain this trapdoor information σ when interacting with the honest verifier. Although at first this may seem to make the modification pointless, this modification is in fact *crucial* to obtaining the zero-knowledge property. The reason is that although the trapdoor information σ is infeasible to obtain in an actual interaction with the verifier, one can construct the protocol such that it will be *easy* to obtain by the simulator. This will allow the simulator to produce a "real-looking" (and in particular accepting) proof, even though it does not get the witness as input.

Protocols following the FLS technique are usually constructed to ensure that using black-box access to the next-message function of the verifier (or in other words, using the power to "rewind" the verifier) it would be easy to obtain the trapdoor information σ (e.g., this is the case in [FS89, RK99, KP00]). Our protocol also uses the FLS technique but with a twist. We construct our protocol in such a way that our trapdoor information σ will simply be the *description* of the verifier's *next-message function* (i.e., the verifier's *code*). Thus a *non-black-box* simulator has (trivially) access to this trapdoor information. Note that because the verifier's next-message function may be a function that is hard to learn (e.g., a pseudorandom function) it may be very hard for a *black-box* simulator to obtain the trapdoor information. Indeed, our protocol will *not* be *black-box* zero-knowledge (as mentioned above every argument system for a non-trivial language satisfying Properties 1–5 *can not* be black-box zero-knowledge).

The techniques of Feige, Lapidot and Shamir [FLS99] allow to use as trapdoor information the witness for any **NP** language. However, it turns out that for our purpose of making the trapdoor information be the verifier's code, this is not sufficient. Loosely speaking, the problem is that the running time of the verifier is not a-priori bounded by any *fixed* polynomial. This problem is similar to a problem that Canetti, Goldreich and Halevi [CGH98] encountered previously, when they tried to construct a counter-example for the Random Oracle Methodology. We solve this problem in a similar way to [CGH98], using *universal arguments*. Using universal arguments we are able to extend the technique of [FLS99] and use as trapdoor information a witness for any $\mathbf{Ntime}(T(n))$ language for some *super-polynomial* function $T(\cdot)$ (e.g., $T(n) = n^{\log\log n}$).

4.1.3 Related Work

Zero-knowledge proofs were introduced by Goldwasser, Micali and Rackoff in [GMR85]. Goldreich, Micali and Wigderson [GMW86] gave a zero-knowledge proof for any language in **NP**, and showed the wide applicability of such proofs to solving protocol problems. *Constant-round* zero-knowledge arguments and proofs for any language in **NP** were first presented by Feige and Shamir [FS89], Brassard , Crépeau and Yung [BCY89], and Goldreich and Kahan [GK96].

All of the above protocols utilized *black-box* simulators (see also [GO87]). Goldreich and Krawczyk [GK90] showed that no language outside of **BPP** has a constant-round Arthur-Merlin zero-knowledge proof or argument system with a *black-box* simulator. They also showed that no language outside of **BPP** has a *general* (i.e., not necessarily Arthur-Merlin) *three-round* zero-knowledge proof or argument system with a *black-box* simulator.

A non-black-box zero-knowledge argument was suggested by Hada and Tanaka [HT99]. However, they used a highly non-standard assumption that in itself was of a strong "reverse-engineering" flavor.

Some of our techniques (i.e., the use of CS proofs for trapdoor information) were first used by Canetti, Goldreich and Halevi [CGH98] for the purpose of constructing cryptographic schemes that are secure in the Random Oracle Model [BR93] but are *insecure* under any implementation of this model. CS proofs were defined and constructed by Kilian [Kil92, Kil95] and Micali [Mic94].

4.1.4 Organization

The construction of our zero-knowledge protocol is described in three stages. In Section 4.2, we construct a zero-knowledge protocol satisfying Properties 2, 4 and 5 of the introduction. That is, we construct a protocol that is constant-round Arthur-Merlin and has a strict polynomial-time simulator. However, it will not be zero-knowledge with respect to auxiliary input (i.e., non-uniform zero-knowledge). Rather, it will only be zero-knowledge with respect to verifiers whose strategy can be implemented by a *uniform* probabilistic polynomial-time Turing machine. In Section 4.2.4 we present an alternative construction for a *uniform-verifier generation protocol*, which is the main component in this construction. This alternative construction is somewhat simpler and more round-efficient.

In Section 4.3, we modify the protocol of Section 4.2 and obtain a protocol that is zero-knowledge with respect to *non-uniform* verifiers. In Section 4.4 we make yet another modification to obtain a protocol that remains zero-knowledge under bounded-concurrent composition.

Section 4.6 contains conclusions and open problems.

4.1.5 Computational Assumptions

Throughout most of this chapter we will make the assumption that there exists a family of hash functions that is collision-resistent against circuits of size $n^{\log n}$ where n is the security parameter. The choice of $n^{\log n}$ is somewhat arbitrary and in fact all these results easily generalize to hold under the weaker assumption that there exist hash functions that are collision-resistent against $f(n)$-sized circuits for some function $f(\cdot)$ that is super-polynomial (i.e., $f(n) = n^{\omega(1)}$) and polynomial-time computable. In Section 4.5 we show that our result holds also under the weaker (and more standard) assumption that that there exist hash functions that are collision-resistent against all polynomial-sized circuits.

4.2 A Uniform Zero-Knowledge Argument

In this section we construct a *constant-round Arthur-Merlin* argument system for **NP** that is zero-knowledge for *uniform* verifiers (i.e., verifiers whose strategy is implemented by a Turing machine without advice). The protocol of this section will utilize a *non-black-box* simulator that runs in *strict* probabilistic polynomial-time. This protocol falls short of satisfying all the properties 1–5 stated in Section 4.1.1 because it is only zero-knowledge against *uniform* verifiers and we do not know whether or not it remains zero-knowledge under bounded concurrent composition. However, it does illustrate the main ideas of our construction.

4.2.1 FLS-type protocols

In our construction, we use a general technique that has been used before in the design of zero-knowledge protocols. We call this technique the *FLS technique*, since it was introduced in a paper by Feige, Lapidot and Shamir [FLS99].

The FLS technique allows to reduce the problem of constructing a zero-knowledge proof (or argument) system to the problem of constructing two simpler objects: a *witness-indistinguishable* (WI) proof/argument system and (what we call here) a *generation protocol*. Witness-indistinguishable proof and argument systems are described in Section 2.3.4. Generation protocols are defined later on (See Definition 4.2.1).

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$)	$w \downarrow$ $x \downarrow$ $\boxed{P}$ $\boxed{V}$
Steps P,V1.x (Generation protocol): Prover and verifier engage in a *generation protocol* GenProt . We denote the transcript of the execution by τ.	$1^n \downarrow$ $\boxed{\textsc{GenProt}}$ $\downarrow \tau$
Steps P,V2.x (WI Proof): Prover proves to verifier using its auxiliary input w via a witness-indistinguishable (WI) proof/argument system that either $x \in L$ or $\tau \in \Lambda$, where Λ is a fixed language (which is part of the protocol's specification).[3]	$w \downarrow$ $x,\tau \downarrow$ $\boxed{WI\text{-}proof \; x \in L \; \textbf{or } \tau \in \Lambda}$ $\downarrow 0/1$
The verifier accepts if the WI proof of the second stage is completed successfully (i.e., if the verifier algorithm for the WI proof accepts).	

The right column contains a schematic description of the protocol as defined in the left column.

Figure 4.1: A generic FLS-type zero-knowledge protocol

We call a zero-knowledge protocol that is constructed using the FLS technique an *FLS-type* protocol. Figure 4.1 describes a generic FLS-type zero-knowledge protocol. The general outline of such a protocol is that when proving some statement of the form "$x \in L$" first the prover and the verifier engage in a generation protocol. Then the prover proves to the verifier using a WI system that *either* the statement "$x \in L$" is true, *or* a different statement about the transcript of the generation protocol is true. To obtain a specific protocol one needs to specify the generation protocol (GenProt) to be used in Steps P,V1.x, the language Λ, and the WI proof (or argument) to be used in Steps P,V2.x. We stress that there are also *black-box* zero-knowledge arguments that use the FLS technique (e.g., [RK99, KP00]).

Note that the generation protocol does *not* take the statement x as an input, and so honest parties do not use x in computing their strategies for the first phase. (Although cheating parties may choose to do so.) Intuitively, one may think of the generation protocol as a game that the prover and verifier play. The prover's objective in this game is to make the transcript τ of the protocol in the language Λ, while the verifier's objective is to ensure that τ will *not* be in Λ. By the way an FLS-type protocol is set up, if the prover "wins" in the generation phase, then he doesn't need to prove that $x \in L$ in the second phase. Thus, for the proof to be sound, it is important that an honest verifier will win this game with high probability, no matter what strategy the prover may use. However, we will need to require some additional properties from the generation protocol in order to make it useful in this setting.[4]

[3]Formally, the prover proves that $\langle x,\tau\rangle \in L'$ where the language L' is defined as follows: $\langle x,\tau\rangle \in L'$ if $x \in L$ or $\tau \in \Lambda$.

[4]In particular, it will have the property, useful for demonstrating that the larger protocol is zero-knowledge, that

Defining generation protocols. We now turn to formally defining what is a generation protocol. Our definition is motivated by our intended application. Therefore, we make requirements from a generation protocol that will ensure that when a generation protocol is plugged into the generic construction of Figure 4.1, the result would be a zero-knowledge proof or argument system. We define a *uniform-verifier* generation protocol since in this section we are only interested in obtaining a protocol that is zero-knowledge against verifiers whose strategy can be implemented by a *uniform* probabilistic polynomial-time Turing machine. The formal definition follows:

Definition 4.2.1 (Uniform-verifier generation protocol). Let GenProt be a two-party protocol where we call one party the *prover* and the other party the *verifier*. Let $\Lambda \subseteq \{0,1\}^*$ be some language in $\mathbf{Ntime}(T(n))$ for some (polynomial-time computable) function $T : \mathbb{N} \to \mathbb{N}$ (e.g., $T(n) = n^{\log\log n}$ or $T(n) = n^3$). We say that GenProt is a *(uniform-verifier) generation protocol* (with respect to the language Λ) if it satisfies the following two requirements:

Soundness (This requirement ensures that the protocol that GenProt will be plugged into will be sound.) Let τ denote the transcript of the execution of GenProt. If the verifier follows its prescribed strategy then, regardless of the prover's (efficient or inefficient) strategy, it holds that $\Pr[\tau \in \Lambda] < \mu(n)$ for some negligible function $\mu : \mathbb{N} \to [0,1]$.

Simulation of uniform verifiers (This requirement ensures that the protocol that GenProt will be plugged into will be zero-knowledge against uniform verifiers.) There exists a *simulator* S_{GenProt} that satisfies the following:

Let V^* be an interactive strategy for the verifier that runs in polynomial-time and can be described using less than $2n$ bits where n is the security parameter. Then on input the description of V^*, S_{GenProt} runs for time polynomial in the running time of V^* and outputs a pair (v, σ) such that:

1. v is computationally indistinguishable from the view of V^* in an execution of GenProt with the prescribed prover algorithm.
2. Let τ denote the transcript that is contained in the view v. Then it holds that $\tau \in \Lambda$ and σ is a witness to this fact. Furthermore, we require that the time to verify that σ is a witness for τ is polynomial in the running time of V^*.[5]

Note that the two requirements together imply that Λ is a hard language. This is because in a real execution with the honest verifier it is almost always the case that the transcript τ is not in Λ while in the computationally indistinguishable simulated execution it is always in Λ. Note also that the simulator S_{GenProt} is given the description of V^* as input and so may possibly make a non-black-box use of this description.

We can now prove the main theorem that we need about the FLS technique:

Theorem 4.2.2. *Let* GenProt *be a generation protocol with respect to an* $\mathbf{Ntime}(T)$ *language* Λ *(where* $T : \mathbb{N} \to \mathbb{N}$ *is a polynomial-time computable function). Let* WIProt*be a WI proof or argument system for* $\mathbf{NP} \cup \mathbf{Ntime}(T)$ *languages. Let* L *be an* $\mathbf{NP}$ *language and let* FLSProt *be the argument for* L *that is the result of plugging in* GenProt *and* WIProt *into the construction of Figure 4.1. Then* FLSProt *is a uniform zero-knowledge argument for* L.

if the prover knows the strategy and random tape of the verifier then he can always win the game.

[5]This requirement is important when considering $\Lambda \in \mathbf{Ntime}(T(\cdot))$ for a super-polynomial function $T(\cdot)$.

Note that we deliberately stated Theorem 4.2.2 in a way that allows to treat in a uniform way both the case that $\Lambda \in \mathbf{NP}$ and the case that $\Lambda \in \mathbf{Ntime}(T) \setminus \mathbf{NP}$ for some super-polynomial function $T(\cdot)$. In the former case it is sufficient to use a standard WI proof system for **NP** such as the one of [FS90]. In the latter case one needs to use a WI *universal argument* (see Section 2.4). Note that previous FLS-type protocols used languages $\Lambda \in \mathbf{NP}$ but we will need to use $\Lambda \in \mathbf{Ntime}(T(\cdot))$ for some super-polynomial $T(\cdot)$.

Proof sketch of Theorem 4.2.2

We only sketch the proof of Theorem 4.2.2 since it will be superseded by a non-uniform analogue (Theorem 4.3.2). To show that FLSProt is a zero-knowledge argument one needs to show three properties: completeness, soundness, and zero-knowledge.

Completeness. Completeness follows from the fact that if the public input x is in L then the statement "$x \in L$ or $\tau \in \Lambda$" is true. Furthermore, the witness w for x can serve as a witness for this statement. Therefore completeness follows from the completeness with efficient prover condition of the WI proof/argument system. Note that since $L \in \mathbf{NP}$, if $x \in L$ then verifying that *either* $x \in L$ or $\tau \in \Lambda$ can be done in non-deterministic polynomial-time, even if deciding Λ takes non-deterministic *super-polynomial* time. This is because the witness w for x is also a witness for the combined statement.

Soundness. Suppose that $x \notin L$. Let τ denote the transcript of the first stage (Steps P,V1.x) of FLSProt. By the soundness property of GenProt with very high probability $\tau \notin \Lambda$. Therefore the combined statement "$x \in L$ or $\tau \in \Lambda$" will be false with very high probability and so the prover will not succeed in convincing the verifier by the soundness of the WI proof/argument system.

Uniform zero-knowledge. To show that FLSProt is zero-knowledge against uniform verifiers one should exhibit a simulator. Algorithm 4.2.3 is such a simulator. The simulator's operation can be summarized as follows: it uses the simulator S_{GenProt} of the generation protocol GenProt to obtain both a simulation v for the first stage along with a witness σ that $\tau \in \Lambda$ where τ is the transcript that v contains. Then, it uses the *honest prover algorithm* of the WI system WIProt to prove the true statement "$x \in L$ or $\tau \in \Lambda$", while using the witness σ as auxiliary input to the prover algorithm of WIProt. The first stage (running the simulator S_{GenProt}) can certainly be done in time that is polynomial in the running time of V^*. The second step (running the honest prover algorithm) can be done in time that is a fixed polynomial in the size of the statement if $\Lambda \in \mathbf{NP}$. However, even if $\Lambda \notin \mathbf{NP}$, this step cam still be performed in time polynomial in the time to verify that σ is a witness that $\tau \in \Lambda$ (using the completeness with efficient property of universal arguments, see Section 2.4). This is polynomial in the running time of V^* by Item 2 in the uniform-verifier simulation condition of GenProt. Item 1 of the uniform-verifier simulation condition of GenProt, along with the witness indistinguishability property of WIProt, ensure that the output of our simulator will indeed be computationally indistinguishable from the view of the verifier in a real interaction.

[6] Note that we do not need to assume that V^* is a completely uniform Turing machine but only that its description is at most n-bits long.

<table>
<tr><td>

Input:

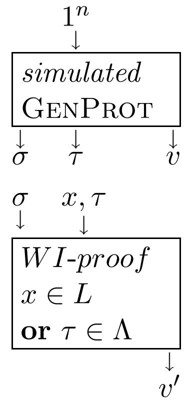
- $x \in \{0,1\}^n$: statement (simulate proof for "$x \in L$")
- V^*: description of the Turing machine of verifier.

</td><td></td></tr>
<tr><td>Let V^{**} denote the verifier V^* with x "hardwired" into it. Note that since V^* is a Turing machine we can assume that the description of V^{**} takes at most $2n$ bits.[6]</td><td></td></tr>
<tr><td>

Simulated Steps P,V1.x (Simulated generation protocol): Let $(v, \sigma) \leftarrow S_{\text{GenProt}}(V^{**})$ where S_{GenProt} is the simulator for the generation protocol GenProt. Let τ denote the transcript contained in the view v. We let V^{***} denote the *residual* verifier V^{**} with the view v hardwired in.

Simulated Steps P,V2.x (Honest WI Proof): Run an execution of WIProt between the verifier V^{***} and the honest prover algorithm for the WI system WIProt the statement proved is "$x \in L$ or $\tau \in \Lambda$" using the witness σ. Let v' denote V^{***}'s view in this execution.

</td><td>



</td></tr>
<tr><td>Output the combined view (v, v') of the two stages</td><td></td></tr>
</table>

This is a non-interactive algorithm. The right side contains a schematic description of the steps simulated in the left side.

Algorithm 4.2.3. A simulator for the FLS-type protocol FLSProt.

Public input: 1^n: security parameter	1^n $\downarrow$ $\boxed{P}$ $\boxed{V}$
Step P1 (Commitment to "junk"): Prover computes $z \leftarrow_{\mathrm{R}} \mathsf{Com}(0^{3n})$ and sends z to the verifier.	$\xrightarrow{z = \mathsf{Com}(0^{3n})}$
Step V2 (Send random string): The verifier selects a string $r \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends it.	$\xleftarrow{r \leftarrow_{\mathrm{R}} \{0,1\}^n}$
The transcript of the protocol is the pair $\tau = (z, r)$.	

Protocol 4.2.4. A uniform-verifier generation protocol

4.2.2 A uniform-verifier generation protocol

Now that we have described the FLS technique we see that to describe our zero-knowledge protocol we should only specify the two components used (i.e., the WI system and generation protocol). Since we want the zero-knowledge to have a constant number of rounds and to be of the Arthur-Merlin type we must ensure that both components are indeed constant-round and Arthur-Merlin. For the WI system we will use the constant-round Arthur-Merlin WI universal argument obtained from Theorem 2.4.4. We will use the universal arguments system to prove membership in $\mathbf{Ntime}(n^{\log\log n})$ languages. Thus, our main challenge is to construct a constant-round Arthur-Merlin generation protocol with respect to some language $\Lambda \in \mathbf{Ntime}(n^{\log\log n})$. We construct such a generation protocol now. We remark that a reader that just want to get the flavor of our techniques may want to look at Section 4.2.4, where we present a somewhat simpler generation protocol. However, the current protocol generalizes more easily to the non-uniform case, which is why we choose to focus on it.

Protocol 4.2.4 is our uniform-verifier generation protocol. It consists of two rounds where in the first message the prover sends a commitment to a "junk" string (i.e., 0^{3n}) and in the second message the verifier sends a random string of length n. However, to fully specify the generation protocol one needs to specify the language Λ, which is what we do next.

Definition of the language Λ. We shall now specify the language Λ. Recall that for a string y, $\mathsf{Com}^{-1}(y)$ denotes the unique x such that y is a commitment to x or $\perp$ if no such x exists. That is $x = \mathsf{Com}^{-1}(y)$ if there exists s such that $\mathsf{Com}(x; s) = y$. We define Λ in the following way: let $\tau = (z, r)$ is in Λ iff on input z, the Turing machine described by $\mathsf{Com}^{-1}(z)$ halts and outputs r within $|r|^{\log\log|r|/5}$ steps.[7] (If $\mathsf{Com}^{-1}(z) = \perp$ or $\mathsf{Com}^{-1}(z)$ does not describe a valid Turing machine then $\tau = (z, r) \notin \Lambda$.) In other words, Λ is defined as follows:

$$\boxed{(z,r) \in \Lambda \iff \Pi(z) \text{ outputs } r \text{ within } |r|^{\log\log|r|/5} \text{ steps, where } \Pi = \mathsf{Com}^{-1}(z)}$$

As a first observation, note that $\Lambda \in \mathbf{Ntime}(n^{\log\log n})$. Indeed, using non-determinism it is possible to obtain $\Pi = \mathsf{Com}^{-1}(z)$ and then we have enough time to simulate the Turing machine described by Π for $n^{\log\log n/5}$ steps.

We can now present the main theorem of this section:

[7] Again, we chose $|r|^{\log\log|r|/5}$ rather arbitrarily. We just need to ensure that Λ will be in $\mathbf{Ntime}(n^{\log\log n})$.

Theorem 4.2.5. *Protocol 4.2.4 is a uniform-verifier generation protocol (as per Definition 4.2.1).*

To prove Theorem 4.2.5, one needs to prove that Protocol 4.2.4 satisfies both the soundness and the uniform-verifier simulation properties. We start with the soundness:

Claim 4.2.5.1. *Let P^* be any (possibly cheating) prover strategy for Protocol 4.2.4. Let τ denote the transcript of P^*'s execution with the honest verifier. Then* $\Pr[\tau \in \Lambda] \leq 2^{-n}$.

Proof. For every first prover message z, we define $f(z)$ to be the output of the Turing machine described by $\mathsf{Com}^{-1}(z)$ on input z after $n^{\log\log n/5}$ steps if $\mathsf{Com}^{-1}(z)$ is a valid Turing machine that on input z halts within this number of steps; otherwise, we define $f(z) = \perp$. For every string z, if $f(z) = \perp$ then $(z, r) \notin \Lambda$ for every r. If $f(z) \neq \perp$, then $(z, r) \in \Lambda$ iff $r = f(z)$. yet, the probability that a random $r \leftarrow_{\mathrm{R}} \{0,1\}^n$ will be equal to $f(z)$ is at most 2^{-n}. Therefore, regardless of the prover's first message the probability that the transcript (z, r) will be in Λ is at most 2^{-n}. □

We now turn to the simulation condition:

Claim 4.2.5.2. *There exists a simulator $S_{\textsc{GenProt}}$ for Protocol 4.2.4 such that for every probabilistic polynomial-time verifier V^* whose description takes at most $2n$ bits, $S_{\textsc{GenProt}}(V^*) = (v, \sigma)$ such that*

1. *v is computationally indistinguishable from V^*'s view in an execution of Protocol 4.2.4.*

2. *σ is a witness that the transcript τ contained in the view v is in Λ. Furthermore, it is possible to verify that σ is such a witness in time that is polynomial in the running time of V^*.*

Proof. Algorithm 4.2.6 is a simulator for Protocol 4.2.4. The output of Algorithm 4.2.6 is a pair (v, σ) such that $v = (s, z)$ and σ contains Π such that $\Pi(z) = V_s^*(z)$ and a witness to the fact that $z = \mathsf{Com}(\Pi)$.

The properties that we require from (v, σ) are:

1. v is computationally indistinguishable from V^*'s view in a real execution. This follows from the fact that PRG is a pseudorandom generator and so its output s is computationally indistinguishable from V^*'s random-tape in a real execution and from the fact that Com is a commitment scheme and so $\mathsf{Com}(\Pi)$ is indistinguishable from $\mathsf{Com}(0^{3n})$.

2. The transcript τ contained in v is in Λ and σ is a witness to this fact. The transcript corresponding to v is $(z, V_s^*(z) = r)$. It is indeed in Λ because $z = \mathsf{Com}(\Pi)$ such that on input z, Π outputs r within a polynomial (and therefore less than $n^{\log\log n/5}$) number of steps. The string σ contains Π and the random coins of the commitment z and so is a witness to this fact. Note that since Π is basically the verifier's strategy V^* with some inputs hardwired in, the fact that σ is a witness for τ can be verified in time that is a fixed polynomial in the running time of V^*.

Note that Algorithm 4.2.6 is a *non-black-box* simulator that takes the description of the verifier as input and uses it in other ways than simply as a black-box or oracle. Note also that it runs in *strict* probabilistic polynomial-time. □

<table>
<tr><td>

Input:

- 1^n: security parameter.
- V^*: description of a probabilistic polynomial-time Turing machine. The length of V^* is at most $2n$.

</td><td></td></tr>
<tr><td>

(Choose randomness for V^*): Let m denote the number of random bits V^* uses. Let $\mathrm{PRG} : \{0,1\}^{n/2} \to \{0,1\}^m$ be a pseudorandom generator. Choose $u \leftarrow_{\mathrm{R}} \{0,1\}^{n/2}$ and let $s = \mathrm{PRG}(u)$. We denote by V^{**} the residual verifier V^* with the randomness s hardwired into it.

</td><td></td></tr>
<tr><td>

Simulated step P1 (Commitment to V^*'s program): Let Π denote the next message algorithm of V^{**}. Note that Π can be described using less than $3n$ bits (the description of V^*, the description of PRG and the seed u). Compute $z \leftarrow_{\mathrm{R}} \mathsf{Com}(\Pi)$.

Simulated Step V1 (Compute V^*'s response): Compute the verifier V^*'s response with randomness s to the message z. That is, $r = \Pi(z)$.

</td><td>

$\xrightarrow{z = \mathsf{Com}(\Pi)}$

$\xleftarrow{r = \Pi(z) = V^{**}(z)}$

</td></tr>
<tr><td>

The output of the simulator is the pair (v, σ) where v is the view (s, z) and σ is the witness that (z, r) is in Λ (i.e., σ contains the program Π and the coins used in computing the commitment z).

</td><td></td></tr>
</table>

Algorithm 4.2.6. A simulator for Protocol 4.2.4.

Public input: 1^n: security parameter	1^n $\downarrow$ $\boxed{P}$ $\boxed{V}$
Step V1 (Send random string): The verifier selects a string $r \leftarrow_{\mathrm{R}} \{0,1\}^{6n}$ and sends it.	$\xleftarrow{r \leftarrow_{\mathrm{R}} \{0,1\}^{6n}}$
The transcript of the protocol is the string r.	

Protocol 4.2.7. An alternative uniform-verifier generation protocol

4.2.3 Summing up

When we plug into the construction of Figure 4.1 our generation protocol (Protocol 4.2.4) and the universal argument system for $\mathbf{Ntime}(n^{\log\log n})$ obtained from Theorem 2.4.3, we obtain a zero-knowledge argument system for **NP**. This system has a constant number of rounds and is of the Arthur-Merlin type. The simulator of this zero-knowledge argument is a *non-black-box* simulator that runs in *strict* probabilistic polynomial-time.

In fact, the protocol we obtain is not just zero-knowledge against *fully uniform* verifiers but even against verifiers that have a *bounded amount of non-uniformity.* That is, verifiers that can be described in $n/2$ bits where n is the security parameter.[8] Note that although the results of Goldreich and Krawczyk [GK90] are stated for non-uniform zero-knowledge, their proofs can be extended for the case of *bounded non-uniformity.*[9] Therefore any constant-round Arthur-Merlin argument (such as ours) for a non-trivial language cannot be *black-box* zero-knowledge. This means that our simulator is *inherently* a non-black-box simulator.

Thus, the protocol of this section is sufficient for the purpose of separating black-box from non-black-box zero-knowledge. However, for other purposes, a uniform zero-knowledge protocol (or even a bounded non-uniform zero-knowledge protocol) is not completely satisfactory. For example, we don't know how to prove a *sequential composition theorem* for uniform or even bounded-non-uniformity zero-knowledge arguments [GK90]. In contrast, such a theorem *is* known to hold for *non-uniform* (a.k.a. *auxiliary input*) zero-knowledge protocols. Thus, for many application it is preferred to have such a protocol. In the next section we show how to modify our construction to obtain a *non-uniform* zero-knowledge argument system for **NP**.

4.2.4 An Alternative Uniform-Verifier Generation Protocol

In this section, we sketch an alternative uniform-verifier generation protocol. This alternative generation protocol, Protocol 4.2.7, has the advantage of being extremely simple and round efficient (consisting of only a single round, in which the verifier sends a random string). However, it has the disadvantage of being harder to generalize to the non-uniform case than Protocol 4.2.4. This protocol is not used in any other place in this work.

Definition of the language Λ. To fully specify Protocol 4.2.7, one should also specify the language Λ. Loosely speaking, we want Λ to be the language of strings with *low Kolmogorov complexity* (i.e., strings can be computed by a Turing machine with small description). However,

[8] The value $n/2$ is quite arbitrary: by "scaling" the security parameter for every polynomial $p(\cdot)$ we can obtain an argument system that is secure against verifiers that can be described using at most $p(n)$ bits.

[9] This holds also for other black-box zero-knowledge lower bounds (e.g., [CKPR01, BL02]).

in order to make Λ decidable in time $n^{\log\log n}$, we will restrict ourselves to machines that on input of size n, halt within $n^{\log\log n/5}$ steps. Formally, Λ is defined as follows:

$$\boxed{r \in \Lambda \iff \exists \text{TM } M \text{ s.t. } |M| < \frac{|r|}{2} \text{ and } M() \text{ outputs } r \text{ within } |r|^{\log\log|r|/5} \text{ steps.}}$$ [10]

We now sketch why Protocol 4.2.7 is indeed a uniform-verifier generation protocol:

Soundness By a simple counting argument, it can be shown that a random string r has high (and in particular higher than $\frac{|r|}{2}$) Kolmogorov complexity, and so r will not be a member of Λ with very high probability.

Simulation of a uniform-verifier Let V^* be a possibly cheating verifier whose strategy can be described in $2n$ bits, and suppose that V^* uses a random tape of size $q(n)$. To simulate the view of V^*, the simulator will use a *pseudorandom generator* $PRG : \{0,1\}^{0.1n} \to \{0,1\}^{q(n)}$, and compute $s = PRG(u)$ where $u \leftarrow_{\mathrm{R}} \{0,1\}^{0.1n}$. It will then let r be V^*'s output on input security parameter 1^n and random tape s. Because r can be computed in polynomial time from a machine whose description is at most the sum of the description of V^*, of PRG, and of u, which is less than $3n$, it follows that not only $r \in \Lambda$ but also the simulator has a *witness* to this fact. Furthermore, r is distributed in a computationally indistinguishable way from the output of V^* in a real interaction.

4.3 Coping with Non-Uniform Verifiers

In this section we construct a non-uniform zero-knowledge argument with the properties of the protocol of Section 4.2. That is, we construct an argument system for **NP** with the following properties:

1. It is zero-knowledge with respect to non-uniform adversaries with auxiliary information.
2. It has a constant number of rounds and negligible soundness error.
3. It is an Arthur-Merlin (public coins) protocol.
4. It has a simulator that runs in *strict* polynomial-time, rather than *expected* polynomial-time.

That is, this protocol satisfies all the properties stated in Section 4.1.1 except for Property 3 (bounded concurrent zero-knowledge). A modification of this protocol that satisfies Property 3 is described in Section 4.4.

4.3.1 FLS'-type protocols

Like the uniform-verifier protocol of Section 4.2, our non-uniform protocol will also use the FLS technique. However, we will need a slight relaxation of the soundness condition of the generation protocol (Definition 4.2.1). This time, we will allow the possibility that the transcript τ is in Λ with non-negligible probability. However, we require that even in this case, it will be *infeasible* to come up with a *witness* that τ is indeed in Λ. Such a generation protocol is sufficient to be plugged in the construction of Figure 4.1, if we use a WI proof (or argument) *of knowledge* in the second

[10] We use $M()$ to denote M executed on the empty input.

stage (Steps P,V2.x), rather than just a proof of membership. In contrast, we will strengthen the simulation condition of Definition 4.2.1 and require simulation even of *non-uniform* verifiers. We will call a protocol that satisfies this modified definition a *non-uniform verifier* generation protocol, although we will usually drop the qualifier and simply use the name *generation protocol* for the non-uniform case.

We call a protocol that uses a generation protocol and a WI proof/argument of knowledge in this way an *FLS'-type* protocol. For completeness, we include a description of FLS'-type protocols in Figure 4.2.

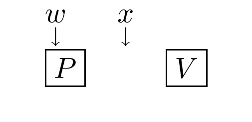
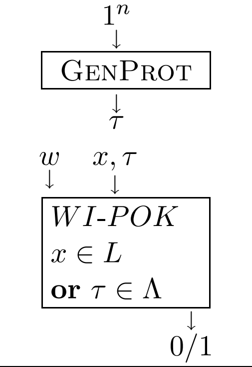
Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$)	
Steps P,V1.x (Generation protocol): Prover and verifier engage in a *non-uniform verifier generation protocol* GenProt . We denote the transcript of the execution by τ. **Steps P,V2.x (WI Proof of knowledge):** Prover proves to verifier using a witness-indistinguishable (WI) proof (or argument) *of knowledge* system that either $x \in L$ or $\tau \in \Lambda$ where Λ is a fixed language, which is part of the protocol's specification. Verifier accepts if proof is completed successfully.	

Figure 4.2: A generic FLS'-type zero-knowledge protocol

The formal definition of non-uniform generation protocols is as follows:

Definition 4.3.1 ((Non-uniform verifier) generation protocol). Let GenProt be a two-party protocol where we call one party the *prover* and the other party the *verifier*. Let $\Lambda \subseteq \{0,1\}^*$ be some language in $\mathbf{Ntime}(T(n))$ for some (polynomial-time computable) function $T : \mathbb{N} \to \mathbb{N}$. We say that GenProt is a *(non-uniform) generation protocol* (with respect to the language Λ) if it satisfies the following two requirements:

Computational soundness For every $T(n)^{O(1)}$-sized (possibly cheating) prover P^* the following holds: let τ denote the transcript of the execution of GenProtbetween P^* and the prescribed verifier. The probability that P^* succeeds in outputting at the end of the interaction a witness that $\tau \in \Lambda$ is negligible.

Simulation of a non-uniform verifier There exists a probabilistic polynomial-time *simulator* $S_{\textsc{GenProt}}$ that satisfies the following:

Let V^* be a polynomial-sized verifier. Then on input the description of V^*, $S_{\textsc{GenProt}}$ outputs a pair (v, σ) such that:

1. v is computationally indistinguishable from the view of V^* in an execution of GenProt with the prescribed prover algorithm.
2. Let τ denote the transcript that is contained in the view v. Then it holds that $\tau \in \Lambda$ and σ is a witness to this fact. Furthermore, we require that the time to verify that σ is a witness for τ is polynomial in the running time of V^*.

Note that the computational soundness requirement refers to $T(n)^{O(1)}$-sized adversaries rather than polynomial-sized adversaries. Indeed if Λ is in $\mathbf{Ntime}(T(n))$ for a super-polynomial function $T(\cdot)$ then it may take a super-polynomial number of steps just to write down a witness. Also note that, unlike Definition 4.2.1, this definition does not imply that deciding Λ is hard but rather only that the search problem corresponding to Λ (of coming up with a witness) is hard.

We now state and prove the non-uniform analogue of Theorem 4.2.2:

Theorem 4.3.2. *Let* GenProt *be a* non-uniform *generation protocol with respect to the* $\mathbf{Ntime}(T)$ *language* Λ *(where* $T : \mathbb{N} \to \mathbb{N}$ *is a polynomial-time computable function). Let* WIProt *be a WI proof or argument system* of knowledge *for* $\mathbf{NP} \cup \mathbf{Ntime}(T)$ *languages. Let* L *be an* $\mathbf{NP}$ *language and let* FLSProt *be the argument for* L *that is the result of plugging in* GenProt *and* WIProt *into the construction of Figure 4.2. Then* FLSProt *is a* non-uniform *zero-knowledge argument for* L*.*

4.3.2 Proof of Theorem 4.3.2

The proof of Theorem 4.3.2 is similar to the proof of Theorem 4.2.2 sketched in Section 4.2.1. To prove that FLSProt is a zero-knowledge argument one needs to prove three properties: completeness, soundness, and zero-knowledge. We will start with the soundness, since this is the main difference between this proof and the proof of Theorem 4.2.2.

Soundness

Let P^{FLSProt} be a polynomial-sized prover for FLSProt and suppose that for some $x \notin L$, the execution of P^{FLSProt} and the honest verifier is accepting with some probability ϵ. We will use P^{FLSProt} to construct a cheating prover P^{GenProt} for the generation protocol GenProt that, after interacting with the honest verifier, will be able to output a witness for the transcript with probability that is polynomially related to ϵ. Thus, if ϵ is non-negligible then this contradicts the computational soundness of GenProt.

The prover P^{GenProt} works in the following way: when interacting with the verifier of protocol GenProt, the prover P^{GenProt} will use the strategy that the prover P^{FLSProt} uses in the first stage of FLSProt on input x. Let τ denote the transcript of this interaction and let v denote the view of the prover in this interaction. After the interaction is completed, the prover P^{GenProt} will compute the *residual* prover P^{FLSProt} with state v. We denote this residual prover by P^{WIProt} since it specifies a strategy for the second stage of FLSProt: the WI proof stage. The prover P^{GenProt} then applies the *knowledge extractor* of the WI system to the P^{WIProt}. Note that this takes $T(n)^{O(1)}$ steps. If the case the extraction is successful P^{GenProt} will obtain a witness to the statement "$x \in L$ or $\tau \in \Lambda$". Since we assume that $x \notin L$ this means that in this case we obtain a witness to the fact that $\tau \in \Lambda$.

We see that to get a contradiction to the computational soundness of GenProt, all we need to show is that the extraction will be successful with probability that is polynomially related to ϵ. (Where ϵ is the overall success of P^{FLSProt} in an execution with the honest verifier.) Indeed, for at least an $\epsilon/2$ of the executions of the first stage, there is an $\epsilon/2$ probability that the second stage will finish successfully. This implies that with $\epsilon/2$ probability, the computed prover P^{WIProt} will have $\epsilon/2$ probability of convincing the verifier that "$x \in L$ or $\tau \in \Lambda$". Yet when this happens, by the proof of knowledge condition of WIProt, the knowledge extractor succeeds in extracting a witness with probability very close to $\epsilon/2$, and this finishes the proof.

Remark 4.3.3. Note that we have actually proven that the resulting zero-knowledge system is not only sound, but actually also satisfies a weak proof of knowledge property (in the sense that if a cheating prover convinces the verifier to accept with some non-negligible probability ϵ then one can extract with probability that is polynomially related to ϵ). We will use this observation in Chapter 5.

Completeness

The proof for completeness follows the proof in the uniform case (See Section 4.2.1). Recall the description of the honest prover's algorithm on Figure 4.2. When given public input x and a witness w to the fact that $x \in L$, the honest prover algorithm for FLSProt runs the honest prover algorithm for GenProt and then runs the honest prover algorithm for the WI system to prove the combined statement "$x \in L$ or $\tau \in \Lambda$" using the witness w. Note that the witness w serves also as a witness for the combined statement, and this witness can be verifier in polynomial-time (since $L \in \mathbf{NP}$). Thus, by the completeness with efficient prover property of the WI system, the honest prover algorithm runs in probabilistic polynomial-time.

Zero-Knowledge

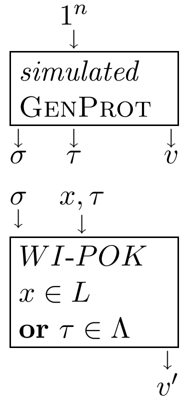
The proof for zero-knowledge also follows the proof in the uniform case (See Section 4.2.1). The simulator for GenProt is Algorithm 4.3.4. The simulator's operation follows the simulator in the uniform case (Algorithm 4.2.3). The simulator uses the simulator S_{GenProt} of the generation protocol GenProt to obtain both a simulation v for the first stage along with a witness σ that is consistent with the transcript that v contains. Then, it uses the *honest prover algorithm* of the WI system WIProt to prove the true statement "$x \in L$ or $\tau \in \Lambda$". It uses the witness σ as auxiliary input to the prover algorithm of WIProt.

What we need to prove is the following claim:

Claim 4.3.4.1. *Let V^* be a polynomial-sized verifier for* FLSProt*. Let $x \in L$. Let (V_R, V'_R) be the random variable that is the view of V^* when interacting with the honest prover on input x (where V_R is the view in the first stage and V'_R is the view in the second stage, R stands for* real *as opposed to* simulated *execution.) Let (V_S, V'_S) be the random variable that is the output of the Algorithm 4.3.4 on input x. Then, (V_R, V'_R) and (V_S, V'_S) are computationally indistinguishable.*

Proof. The proof follows from an hybrid argument. We will prove that both distributions are computationally indistinguishable from the hybrid distribution (V_S, V'_R) where V_S represents the simulation of the first stage and V'_R represents the real execution of the second stage. That is, (V_S, V'_R) is the output of a "hybrid simulator" that uses the simulator of GenProt in the first stage but uses the witness w for x (instead of the witness σ for τ) as input to the WI prover algorithm in the second stage.

- The distribution (V_S, V'_R) is computationally indistinguishable from (V_S, V'_S) due to the WI property of the WI system WIProt.
  Indeed, if there there is an algorithm D that distinguishes between these two distributions with probability ϵ then in particular there must exist a particular view v for the first stage such that D distinguishes between (V_S, V'_R) and (V_S, V'_S) conditioned on $V_S = v$ with probability ϵ. Let τ be the transcript contained in v and let σ be the witness for τ as provided by the simulator. Let D_v be the distinguisher D with v hardwired as its first input and let V^*_v be the residual verifier V^* with the state v hardwired. Then, D_v can distinguish between an

Input: • $x \in \{0,1\}^n$: statement (simulate proof for "$x \in L$") • V^*: description of a polynomial-sized verifier.	
Let V^{**} denote the verifier V^* with x "hardwired" into it.	
Simulated Steps P,V1.x (Simulated generation protocol): Let $(v, \sigma) \leftarrow S_{\textsc{GenProt}}(V^{**})$ where $S_{\textsc{GenProt}}$ is the simulator for the generation protocol GenProt. Let τ denote the transcript contained in the view v. We let V^{***} denote the *residual* verifier V^{**} with the view v hardwired in. **Simulated Steps P,V2.x (Honest WI Proof of knowledge):** Run an execution of WIProt between the verifier V^{***} and the honest prover algorithm for the WI system WIProt the statement proved is "$x \in L$ or $\tau \in \Lambda$" using the witness σ. Let v' denote V^{***}'s view in this execution.	
Output the combined view (v, v') of the two stages	

Algorithm 4.3.4. A simulator for the FLS'-type protocol FLSProt.

interaction of V_v^* and the honest prover of the WI system that uses σ as auxiliary input and an interaction of V_v^* and the honest prover of the WI system that uses w (the witness for x) as auxiliary input with probability ϵ. Thus, ϵ is negligible by the WI condition of WIProt.

- The distribution (V_S, V_R') is computationally indistinguishable from (V_R, V_R') due to the simulation condition of the generation protocol GenProt.
  Indeed, suppose that there is an algorithm D that distinguishes between these two distributions with probability ϵ. Then we can construct a distinguisher D' to contradict the simulation condition of the generation protocol in the following way. The distinguisher D' has the witness w for x hardwired in. When it gets a string v as input, it runs the honest prover algorithm of the WI proof for the statement "$x \in L$ or $\tau \in \Lambda$" (where τ is the transcript contained in v) using the witness y. It plays the part of the verifier using the residual verifier V^* with state v. Let v' denote the view of the verifier in the WI proof. The distinguisher D' returns $D(v, v')$. We see that D' distinguishes between the view of V^* in a real interaction of GenProt and the output of $S_{\textsc{GenProt}}(V^*)$ with probability ϵ and so ϵ is negligible by the simulation condition of GenProt.

□

4.3.3 A Non-Uniform Verifier Generation Protocol

Once we have Theorem 4.3.2, all that is left to do is to construct a *non-uniform* verifier generation protocol with respect to some $\mathbf{Ntime}(n^{\log\log n})$ language Λ. Our non-uniform verifier generation protocol will be based on the uniform-verifier generation protocol of the previous section (Protocol 4.2.4).

Public input: 1^n: security parameter	$1^n \downarrow$ $\boxed{P}$ $\boxed{V}$
Step V1 (Choose hash-function): Verifier chooses a random hash function $h \leftarrow_R \mathcal{H}_n$ and sends h to prover.	$\xleftarrow{h \leftarrow_R \mathcal{H}_n}$
Step P2 (Commitment to hash of "junk"): Prover computes $z \leftarrow_R \mathsf{Com}(h(0^n))$ and sends z to verifier.	$\xrightarrow{z = \mathsf{Com}(h(0^n))}$
Step V3 (Send random string): The verifier selects a string $r \leftarrow_R \{0,1\}^n$ and sends it.	$\xleftarrow{r \leftarrow_R \{0,1\}^n}$
The transcript of the protocol is the pair $\tau = (h, z, r)$.	

Protocol 4.3.5. A non-uniform verifier generation protocol

Recall how we proved that Protocol 4.2.4 satisfies the uniform simulation condition. The first message of the protocol was supposed to be a commitment to 0^{3n}. However, the *simulator* (Algorithm 4.2.6) simulated this message by commitment z to the *next-message function* of the verifier. This may be problematic if we try to use the same protocol and simulator in the non-uniform setting. The problem is that since a commitment scheme is statistically binding, then it necessarily holds that the length of the commitment z will be longer than the length of the description of the next message function. However, once we consider *non-uniform* verifiers then we must allow for the next-message function's description to be of any polynomial length, and in particular it may be larger than the communication complexity of our protocol. The solution is quite simple. Instead of using a statistically binding commitment scheme, we will use a *computationally binding* commitment scheme. A computationally binding commitment scheme that allows to commit to messages that are longer than its output can be constructed by composing a standard, statistically binding commitment scheme, with a collision-resistent hash function. One can see why we had to relax the soundness condition of the definition of a generation protocol: once we use a computationally binding commitment scheme, we will only be able to prove that are protocol is *computationally sound*. This is the intuition that we follow in both the construction of the generation protocol and the definition of the corresponding language Λ. Protocol 4.3.5 is our generation protocol.

Definition of the language Λ. We define the language Λ as follows: $\tau = (h, z, r)$ is in Λ if there exists a program Π such that $z = \mathsf{Com}(h(\Pi))$ and $\Pi(z)$ outputs r within $|r|^{\log\log|r|/5}$ steps. This can be verified in $\mathbf{Ntime}(n^{\log\log n/5})$. A *witness* that $(h, z, r) \in \Lambda$ is a couple (Π, s) such that $z = \mathsf{Com}(h(\Pi); s)$ and $\Pi(z)$ outputs r within $|r|^{\log\log|r|/5}$ steps. Note that it may be the case that Λ is easy to *decide* (in fact it may be that $\Lambda = \{0,1\}^*$) but the soundness condition of Definition 4.3.1 refers only to the infeasibility of coming up with a witness.

Theorem 4.3.6. *Protocol 4.3.5 is a (non-uniform verifier) generation protocol with respect to the language Λ (as per Definition 4.3.1).*

4.3.4 Proof of Theorem 4.3.6

To prove that Protocol 4.3.5 meets Definition 4.3.1 we need to show that it satisfies two properties: computational soundness and non-uniform simulation.

Computational Soundness

Let P^* be a $n^{O(\log\log n)}$-sized prover strategy for Protocol 4.3.5. Let τ denote the transcript of P^*'s execution with the honest verifier. We claim that the probability that P^* writes a witness that $\tau \in \Lambda$ on its auxiliary tape is negligible.

Indeed, suppose otherwise that P^* manages to output a witness with non-negligible probability ϵ. Then, for at least an $\epsilon/2$ fraction of the $h \in \mathcal{H}_n$, it holds that P^* manages to output a witness for the transcript starting with h with probability $\epsilon/2$. Fix such an $h \in \mathcal{H}_n$. Since P^* is a non-uniform algorithm, we can assume without loss of generality that P^* is deterministic. Thus the message z, which is the prover P^*'s response to h, is also fixed. By our assumption, if we choose $r \leftarrow_{\mathrm{R}} \{0,1\}^n$, then with probability $\epsilon/2$ the prover will be able to output a program Π such that z is a commitment to $h(\Pi)$ and $\Pi(z) = r$ (within $|r|^{\log\log|r|/5}$ steps). This means that if we choose two independent $r, r' \leftarrow_{\mathrm{R}} \{0,1\}^n$, then with probability $\epsilon^2/4$, we obtain two programs Π, Π' such that z is a commitment to both $h(\Pi)$ and $h(\Pi')$ and $\Pi(z) = r$, $\Pi'(z) = r'$. Since $\mathsf{Com}(\cdot)$ is a statistically binding commitment scheme, it follows that $h(\Pi) = h(\Pi')$. Yet, since we can assume that $r \neq r'$ (as this holds with $1 - 2^{-n}$ probability), it follows that $\Pi(z) \neq \Pi'(z)$ and so Π and Π' are different programs. This means that Π and Π' are a *collision* for h. This means that we have a $n^{O(\log\log n)}$-sized algorithm that, for an $\epsilon/2$ fraction of $h \in \mathcal{H}_n$, obtains a collision for h with probability $O(\epsilon^2)$. This contradicts the collision-resistence against $n^{\log n}$-sized adversaries of the family $\mathcal{H}_n$.

Simulation of a non-uniform verifier

The proof that Protocol 4.3.5 satisfies the simulation requirement is quite similar to its uniform analog (Claim 4.2.5.2). What we need to show is that there exists a simulator $S_{\textsc{GenProt}}$ for Protocol 4.3.5 such that for every polynomial-sized verifier V^*, $S_{\textsc{GenProt}}(V^*)$ outputs a pair (v, σ) such that v is computationally indistinguishable from V^*'s view and σ is a witness that the transcript τ compatible with v is in Λ.

Algorithm 4.3.7 is a simulator for Protocol 4.3.5. As we can see, when simulating an execution with transcript (h, z, r), the output of Algorithm 4.3.7 is a pair (z, σ) and $\sigma = (\Pi, s)$ is such that $z = \mathsf{Com}(h(\Pi); s)$. The two properties that we require from (z, σ) are:

1. z is computationally indistinguishable from V^*'s view in a real execution. In a real execution the verifier sees a single message which is $\mathsf{Com}(h(0^n))$. The message z is equal to $\mathsf{Com}(h(\Pi))$. Thus this property follows immediately from the computational hiding property of the commitment scheme Com.

2. The transcript τ corresponding to v is in Λ and σ is a witness to this fact. The transcript corresponding to v is (h, z, r) (where $h = V^*()$ and $r = H(z)$). The pair $\sigma = (\Pi, s)$ is indeed a witness that $(h, z, r) \in \Lambda$ since $z = \mathsf{Com}(h(\Pi); s)$ and $\Pi(z)$ outputs r in a polynomial (and so less than $|r|^{\log\log|r|/5}$) number of steps. Note that indeed the time to verify that σ is a witness for τ is polynomial in the running time of V^*.

Note that Algorithm 4.3.7, like Algorithm 4.2.6, is a *non-black-box* simulator. Note also that it runs in *strict* probabilistic polynomial-time.

This finishes the proof of Theorem 4.3.6. By using this generation protocol in Theorem 4.3.2 we obtain a non-uniform zero-knowledge argument with all the desired properties.

<table>
<tr><td>Input:
1^n: security parameter.
V^*: a polynomial-sized circuit (without loss of generality V^* is deterministic).</td><td></td></tr>
<tr><td>Simulated Step V1 (Choose hash-function): Compute h: the V^*'s first message.
Simulated step P2 (Commitment to V^*'s program): Let Π denote the next message algorithm of V^*. Compute $z = \mathsf{Com}(h(\Pi); s)$ where $s \leftarrow_{\mathrm{R}} \{0,1\}^{poly(n)}$ are coins chosen for the commitment scheme.
Simulated Step V3 (Compute V^*'s response): Compute the verifier V^*'s response to the message z. That is, $r = \Pi(z)$.</td><td>$\xleftarrow{h = V^*()}$
$\xrightarrow{z = \mathsf{Com}(h(\Pi))}$
$\xleftarrow{r = \Pi(z) = V^*(z)}$</td></tr>
<tr><td>The output of the simulator is the pair (z, σ) where z is the simulated verifier's view and $\sigma = (\Pi, s)$ is the witness that (h, z, r) is in Λ.</td><td></td></tr>
</table>

Algorithm 4.3.7. A simulator for Protocol 4.3.5.

4.4 Achieving Bounded-Concurrent Zero-Knowledge

The condition that a proof/argument system is zero-knowledge guarantees the prover that a possibly malicious verifier will not be able to gain any new knowledge about the statement that is being proved. However, somewhat surprisingly, it turns out that this may *not* be the case if the prover is proving two or more related statements at the same time [GK90]. To guarantee security for the prover in this (quite realistic) setting, one needs a stronger form of zero-knowledge. This stronger form, called *concurrent zero-knowledge*, was introduced by Dwork, Naor and Sahai [DNS98]. Loosely speaking, a protocol is concurrent zero-knowledge if it remains zero-knowledge even when any polynomial number of possibly related statements are being proved simultaneously, with the scheduling of messages chosen (possibly in a malicious way) by the verifier.

Dwork *et al.*[DNS98] constructed a concurrent zero-knowledge argument for **NP** in the *timing* model, which is a model that assumes a some known time bounds on the delivery of messages in the communication network (See also [Gol01a]). Richardson and Kilian [RK99] were the first to construct a concurrent zero-knowledge argument for **NP** in the standard (pure asynchronous) model. Their protocol used a polynomial number of rounds. This was later improved by Kilian and Petrank to a polylogarithmic number of rounds [KP00] and further improved by Prabhakaran, Rosen and Sahai to a slightly super-logarithmic number of rounds [PRS92]. This is essentially the best one can obtain using *black-box* simulation as shown by Canetti, Kilian, Petrank and Rosen [CKPR01] (improving on [KPR98] and [Ros00]).

In this section, we show how to modify the zero-knowledge protocol of the last section as to obtain a protocol that is *bounded concurrent* zero-knowledge. A zero-knowledge protocol is bounded-concurrent zero-knowledge if it remains zero-knowledge when executed up to n times concurrently, where n is the security parameter. Since the security parameter can be "scaled", this means that for every *fixed* polynomial $p(\cdot)$, we can construct a protocol that remains zero-knowledge when executed $p(n)$ times. However, this protocol will depend on $p(\cdot)$ and in particular it will have communication complexity greater than $p(n)$. This is in contrast with the notion of (unbounded) concurrent zero-knowledge described above, where there is a single protocol that remains zero-knowledge when executed $p(n)$ times for *every* polynomial $p(\cdot)$. We stress that the negative results of [CKPR01] regarding concurrent black-box zero-knowledge hold also for *bounded* concurrent zero-knowledge. In particular, there does not exist a constant-round bounded-concurrent zero-knowledge proof/argument (for a non-trivial language) that utilizes a *black-box* simulator.

Formally, we define bounded-concurrent zero-knowledge as follows:

Definition 4.4.1 (Concurrent execution). Let (P, V) be a two-party protocol. Let V^* be an interactive machine. Let $\{(a_i, b_i)\}_{i=1}^t$ be a set of t inputs to the protocol (P, V). A *t-times concurrent execution* of (P, V) coordinated by V^* on inputs $\{(a_i, b_i)\}_{i=1}^t$ is the following experiment:

1. Run t independent copies of P with the i^{th} copy getting a_i as input.
2. Provide V^* with the $b_1, \ldots, b_t$.
3. On each step V^* outputs a message (i, m). The i^{th} copy of P is given with the message m. The verifier V^* is given the prover's response.

Definition 4.4.2 (Bounded-concurrent zero-knowledge). Let (P, V) be an interactive proof *orargument* system for a language $L = L(R)$. We say that (P, V) is *bounded-concurrent zero-knowledge* if there exists a probabilistic polynomial-time algorithm S such that for every polynomial-sized V^*, and every list $\{(x_i, y_i)\}_{i=1}^n$ such that $(x_i, y_i) \in R$, the following two random variables are computationally indistinguishable:

1. The view of V^* in an n-times concurrent execution of (P, V) with inputs $\{(x_i, y_i)\}_{i=1}^n$.
2. $S(V^*, x_1, \ldots, x_n)$

Our protocol will be an FLS'-type protocol, and will use a (non-uniform verifier) generation protocol which is very similar to Protocol 4.3.5. In fact, the only difference between the generation protocol of this section and Protocol 4.3.5 will be that we will use a longer string r as the verifier's message (Step V2). That is, we will use $r \leftarrow_{\mathrm{R}} \{0,1\}^{n^4}$ rather than $r \leftarrow_{\mathrm{R}} \{0,1\}^n$.[11] Protocol 4.4.3 is the modified generation protocol.

Definition of the language Λ. We use a somewhat different language Λ that the one used in the previous section. We define Λ as follows: $\tau = (h, z, r)$ is in Λ iff there exists a program Π such that $z = \mathsf{Com}(h(\Pi))$ and *there exists a string* y such that $|y| \leq |r|/2$ and $\Pi(z, y)$ outputs r within $|r|^{\log\log|r|/5}$ steps. This can be verified in $\mathbf{Ntime}(n^{\log\log n/5})$. A witness that $(h, z, r) \in \Lambda$ is a triple (Π, s, y) such that $z = \mathsf{Com}(\Pi; s)$, $|y| \leq |r|/2$ and $\Pi(z, y)$ outputs r within $|r|^{\log\log|r|/5}$ steps. That is, we've changed the language so that the committed program Π that outputs r can get not only z as input but is also allowed an additional input y, as long as it is not too long (i.e., as long as $|y| \leq |r|/2$).

The following theorem states that the modified protocol is still a generation protocol:

Theorem 4.4.4. *Protocol 4.4.3 is a (non-uniform verifier) generation protocol with respect to Λ*

[11]The value n^4 is also somewhat arbitrary. We have not tried to optimize the relation between the communication complexity and the number of concurrent sessions.

Public input: 1^n: security parameter	1^n $\downarrow$ $\boxed{P}$ $\boxed{V}$
Step V1 (Choose hash-function): Verifier chooses a random hash function $h \leftarrow_{\mathrm{R}} \mathcal{H}_n$ and sends h to prover.	$\xleftarrow{h \leftarrow_{\mathrm{R}} \mathcal{H}_n}$
Step P2 (Commitment to hash of"junk"): Prover computes $z \leftarrow_{\mathrm{R}} \mathsf{Com}(h(0^n))$ and sends z to verifier.	$\xrightarrow{z = \mathsf{Com}(h(0^n)}$
Step V3 (Send long random string): The verifier selects a string $r \leftarrow_{\mathrm{R}} \{0,1\}^{n^4}$ and sends it.	$\xleftarrow{r \leftarrow_{\mathrm{R}} \{0,1\}^{n^4}}$
The transcript of the protocol is the pair $\tau = (h, z, r)$.	

Protocol 4.4.3. A generation protocol for bounded concurrent zero-knowledge.

Proof Sketch: We only sketch the proof because it is almost identical to the previous section (the proof of Theorem 4.3.6 in Section 4.3.4). The fact that the message r is longer does not change anything in the proof, and so we only need to see that the modification to Λ did no harm.

Indeed, the proof of the non-uniform simulation requirement is unchanged, since the simulator presented in the proof of Theorem 4.3.6 (Algorithm 4.3.7) is also a simulator for Protocol 4.4.3. The witness this simulator outputs is a valid witness also for the modified language of this section (it simply uses the empty word for the string y).

The proof of the computational soundness is slightly changed but still works. Recall that the proof there (in Section 4.3.4) relied on converting a cheating prover into an algorithm to find collision for the hash functions. We used there the following observation: if for some value z ,Π is a program such that $\Pi(z) = r$ and we choose a random $r' \leftarrow_{\mathrm{R}} \{0,1\}^n$ and obtain with probability ϵ a program Π' such that $\Pi'(z) = r'$, then Π will be different from Π' with probability at least $\epsilon - 2^{-n}$. We used this observation to show that we can use a cheating prover to obtain a collision pair Π and Π' for the hash function.

The key observation we need to use now is the following: if for some value z, Π is a program such that $\exists_{y \in \{0,1\}^{m/2}} \Pi(z, y) = r$ and we choose a random $r' \leftarrow_{\mathrm{R}} \{0,1\}^m$ and obtain with probability ϵ a program Π' such that $\exists_{y' \in \{0,1\}^{m/2}} \Pi'(z, y') = r'$, then Π will be different from Π' with probability at least $\epsilon - 2^{-m/2}$. This is because if $\Pi' = \Pi$ then it must hold that $r' \in \Pi(z, \{0,1\}^{m/2})$ which happens with probability at most $2^{-m/2}$. Note that in our case $m = n^4$ and so $2^{-m/2}$ is a negligible quantity.

Using this observation, the proof of the soundness property follows the proof of Section 4.3.4. $\square$

4.4.1 The Zero-Knowledge Argument

Our bounded-concurrent zero-knowledge protocol for **NP** is Protocol 4.4.5. It is constructed by plugging in the generation protocol of the previous section (Protocol 4.4.3) and the WI universal argument system of Theorem 2.4.4 using the FLS' paradigm (see Figure 4.2).

By the results of the previous sections, it satisfies the completeness, soundness and (standalone) non-uniform zero-knowledge properties. It is also clearly a constant-round Arthur-Merlin protocol. Thus, all that remains is to prove that it remains zero-knowledge under bounded concurrent composition. This is what we do in this section. Thus, our main theorem is the following:

Theorem 4.4.6. *Protocol 4.4.5 is bounded-concurrent zero-knowledge.*

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$)	$w \downarrow$ $\boxed{P}$ $\quad$ $x \downarrow$ $\boxed{V}$
Steps P,V1.x: generation protocol **Step V1.1 (Choose hash-function):** Verifier chooses a random hash function $h \leftarrow_{\mathrm{R}} \mathcal{H}_n$ and sends h to prover. **Step P1.2 (Commitment to hash of"junk"):** Prover computes $z \leftarrow_{\mathrm{R}} \mathsf{Com}(h(0^n))$ and sends z to verifier. *(Short message.)* **Step V1.3 (Send long random string):** The verifier selects a string $r \leftarrow_{\mathrm{R}} \{0,1\}^{n^4}$ and sends it. The transcript of this stage is $\tau = (h, z, r)$.	$\xleftarrow{h \leftarrow_{\mathrm{R}} \mathcal{H}_n}$ $\xrightarrow{z = \mathsf{Com}(h(0^n))}$ $\xleftarrow{r \leftarrow_{\mathrm{R}} \{0,1\}^{n^4}}$
Steps P,V2.1.x (WI universal argument): Prover proves to verifier using a WI universal argument that either $x \in L$ or $\tau \in \Lambda$. *All prover's messages here are short.*	$w \downarrow$ $\quad x, \tau \downarrow$ *WI-UARG* $x \in L$ **or** $\tau \in \Lambda$ $\downarrow$ 0/1

Protocol 4.4.5. A bounded-concurrent zero-knowledge protocol

4.4.2 Proof of Theorem 4.4.6

Our proof that it is bounded concurrent zero-knowledge will not be as modular as our previous proofs (see also Remark 4.4.8). That is, we will argue about the entire zero-knowledge argument as a whole, rather than proving statements about its components.

Message lengths. We will need to consider the lengths of the messages sent by the prover. We call messages that of length less than n^2 bits "short". We will make essential use of the observation that all the prover's messages of Protocol 4.4.5 are "short". (Note that this is not the case for the verifier's messages since the message r of Step V1.3 is of length n^4.) In the first stage, we can assume that the prover's message in Step PV1.2 (the commitment $z = \mathsf{Com}(h(0^n))$) is short, because there are constructions of commitment schemes such that the commitment to a message of length n is at most n^2. In the second stage we will use the fact that by Theorem 2.4.4 for every $\epsilon > 0$ there exists a witness-indistinguishable universal argument system with communication complexity m^ϵ where m is the instance length. Therefore, we can ensure that all prover's messages in the second stage will be of length at most n^2. (Even though the length of the statement proven at this stage is more than n^4.)

Overview of the simulator

To prove that Protocol 4.4.5 is bounded-concurrent zero-knowledge, one needs to describe a *simulator* for the protocol, and then analyze its output. We will start with an overview of the simulator's operation, and then provide a detailed description and analysis.

Let V^* be a polynomial-sized algorithm describing the strategy of a verifier in an n-times concurrent execution of Protocol 4.4.5. Recall than an n-time concurrent execution of a protocol involves an execution of n sessions, which are interleaved in a way chosen by the verifier. We denote the j^{th} overall *prover message* (in the entire execution) by m_j. Each message m_j belongs to some session i (where $1 \leq i \leq n$). Because the execution is concurrent it does not necessarily hold that the messages of a particular session are consecutive, but they are always ordered (e.g., the first message of session i always comes before the second message of session i). The *view* of the verifier in the execution is the sequence $(m_1, \ldots, m_{cn})$, where c is the (constant) number of prover messages in Protocol 4.4.5. The object of our simulator is to generate a sequence $(m_1, \ldots, m_{cn})$ that is indistinguishable from the view of the verifier in a real execution.

Our simulator will generate this sequence incrementally. That is, we will generate the message m_j only after we generated $(m_1, \ldots, m_{j-1})$ and once we generated a message we will not "go back" and change it. Therefore, it is possible (and useful) to think of our simulator as interacting with the verifier V^*. Of course, the difference between this interaction and a real interaction is that our simulator has the "unfair" advantage of knowing V^*'s code.

Notation. Throughout this section we will use i to index a session (i.e., $1 \leq i \leq n$) and j and k to index an overall prover message (i.e., $1 \leq j, k \leq cn$). We will use subscript to denote the overall index of a message (e.g., m_j) and use parenthesized superscript to denote the session that a message belongs to (e.g., $r^{(i)}$). We will sometimes drop the session number when it is clear from the context. We will sometime identify a prover or verifier message not by its overall index, but rather by its session number and step number in Protocol 4.4.5. Thus we will say statements like "let $r = r^{(i)}$ denote the verifier message of Step V.1.3 of the i^{th} session".

The naive simulator. The naive first attempt at a simulator would be to try to invoke the standalone simulator of Protocol 4.4.5 n times independently. Let us see exactly where this naive attempt fails. Suppose that we have to compute the k^{th} prover message which is Step P1.2 of some session i. On the previous step (Step V1.1), the verifier sent a hash function $h = h^{(i)}$ and on Step P1.2 we need to compute a commitment $z = z^{(i)}$ to $h(\Pi)$ where $\Pi = \Pi^{(i)}$ is some program. If we follow the instruction of the standalone simulator, then we will let Π be simply the program of the residual verifier V^* at this point. Suppose now that the verifier decides to schedule at this point some other sessions (i.e., different than i). That is, the next messages $m_{k+1}, \ldots, m_{j-1}$ the verifier receives are part of sessions other than i (where m_j is the next prover message after m_k that belongs to the i^{th} session). Suppose that when the verifier sends the string $r = r^{(i)}$ corresponding to Step V1.3 of the i^{th} session, the verifier computes r as a function of the messages $m_{k+1}, \ldots, m_{j-1}$ it received. Since Π was the residual verifier at point k, it is *not* the case that $\Pi(z) = r$. Thus it will not hold that $(h, z, r) \in \Lambda$ and we see that simply running the independent simulator will fail.

However, not all is lost: indeed it *is* the case that $r = \Pi(z, m_{k+1}, \ldots, m_{j-1})$. That is, r is V^*'s response after it receives the messages $m_1, \ldots, m_{j-1}$ where $m_1, \ldots, m_{k-1}$ were already part of Π's description, and $m_k = z$. The crucial observation here is that since all prover's messages are "short" (of length at most n^2), and since there are at most cn messages (where c is the constant number of rounds in Protocol 4.4.5) it holds that

$$|m_{k+1}| + \ldots + |m_{j-1}| \leq O(n^3) < \frac{n^4}{2} = \frac{|r|}{2}$$

Yet this means that we have a witness to the fact that the transcript (h, z, r) is in Λ. This is because, under the definition of Λ in this section, we don't need to show that $\Pi(z) = r$ but rather only to show that $\Pi(z, y) = r$ for *some short string* y (i.e. for y such that $|y| < |r|/2$).

Therefore, we can continue the simulation at this point, and simply use in the session the honest prover strategy for the WI universal argument, with Π, $(m_{k+1},\dots,m_{j-1})$ and the coins used in computing the commitment z.

Actual description of the simulator

Our simulator's operation follows the above description. We now turn to formally describing the simulator algorithm:

Algorithm $\mathcal{S}im$:

Input:

- $x_1,\dots,x_n$: the statement to be proved in the i^{th} session is that $x_i \in L$.
- V^*: description of polynomial-sized verifier coordinating an n-time concurrent execution.

Initialization: The simulator will construct the a table $\mathcal{A}$ of length n. Initially the table $\mathcal{A}$ will be empty. We will maintain the invariant that after we simulated Step P1.2 of the i^{th} session and computed a message $z = z^{(i)}$, $\mathcal{A}[i]$ will contain a pair $(\Pi, s) = (\Pi^{(i)}, s^{(i)})$ such that $z = \mathsf{Com}(h(\Pi; s)$ where $h = h^{(i)}$ is the hash function chosen by the verifier in Step V1.1 of the i^{th} session. After the simulator obtains the message $r = r^{(i)}$ of Step V1.3 from the verifier, it will add to $\mathcal{A}[i]$ a string $y = y^{(i)}$ of length less than $n^4/2$ such that $\Pi(z, y) = r$. That is, at this point $\mathcal{A}[i]$ contains a witness (Π, s, y) for the fact that $(h, z, r) \in \Lambda$ where $h = h^{(i)}$,$z = z^{(i)}$ and $r = r^{(i)}$ are respectively the messages of Steps V1.1, P1.2, and V1.3 of the simulated i^{th} session.

Simulating each step: For $j = 1,\dots,cn$ the simulator computes the j^{th} simulated prover message m_j in the following way:

Feed the previously computed messages $(m_1,\dots,m_{j-1})$ to V^* and obtain the j^{th} verifier message (i, m) (where i is the session the verifier's message is intended to). Compute the message m_j according to the current step in the simulated proof of the i^{th} session:

Step P1.2 - Commitment to program If the verifier's message is for Step V1.1 of the i^{th} session, do the following:

- Let $h = h^{(i)}$ denote the verifier's message.
- Compute the description of the following program Π:

  $\mathbf{\Pi}(\mathbf{z}, \mathbf{y})$ returns $V^*(m_1,\dots,m_{j-1}, z, y)$.

  (Note that the values $m_1,\dots,m_{j-1}$ were previously computed.)
- $z = \mathsf{Com}(h(\Pi); s)$ where s is the randomness for the commitment.
- Store Π and s in $\mathcal{A}[i]$.
- The j^{th} message m_j will be z

Receiving message of Step V1.3 If the verifier's message was for Step V1.3 of the i^{th} message then do as follows:

- Let $r = r^{(i)}$ denote the verifier's message. Note that $r = V^*(m_1,\dots,m_{j-1})$.

- Let k denote the overall index of prover's message in Step P1.2 of the same session. That is, m_k was the message z of the same session.
- Let $y = (y_1, \ldots, y_{j-k-1})$ denote the sequence $(m_{k+1}, \ldots, m_{j-1})$. Note that since $j \leq cn$ we have that $|y| \leq cn \cdot 10n^2$ and so $|y| < n^4/2$ (for sufficiently large n).
- Add y to the cell $\mathcal{A}[i]$. Note that $\mathcal{A}[i]$ already contains (Π, s) such that $z = \mathsf{Com}(\Pi; s)$ and $\Pi(z, y) = r$.

Steps P2.x - WI UARGS In these steps we simply follow the honest prover strategy of the WI universal arguments to prove that either $x_i \in L$ or the transcript $\tau = \tau^{(i)}$ of the first stage of the i^{th} session is in Λ. Note that we can use $\mathcal{A}[i]$ as a witness that indeed the transcript of the first stage is in Λ.

Analysis

The theorem we need to prove is the following:

Theorem 4.4.7. *For every polynomial sized verifier v^* and sequence $\{(x_i, y_i)\}_{i=1}^n$ such that y_i is a witness that $x_i \in L$, the following two random variables are computationally indistinguishable:*

- *The view of V^* in an n-times concurrent execution of Protocol 4.4.5 on inputs $\{(x_i, y_i)\}_{i=1}^n$. We denote this random variable by X.*
- *The output of Algorithm $\mathcal{S}im$ (of Section 4.4.2) on input V^* and $(x_1, \ldots, x_n)$. We denote this random variable by Y.*

Proof. The proof is actually not complicated. We will use the hybrid argument to show that X and Y are computationally indistinguishable. Let $\mathcal{S}im'$ be an algorithm that on input $V^*, \{(x_i, y_i)\}_{i=1}^n$ follows the same strategy as the simulator $\mathcal{S}im$ on input V^* and $(x_1, \ldots, x_n)$, except that when simulating the steps of the WI universal argument (Steps P2.x) in the i^{th} session it will provide y_i as input the honest prover algorithm. Let Z denote the output of $\mathcal{S}im'$ on input V^* and $V^*, \{(x_i, y_i)\}_{i=1}^n$. We will prove the theorem by showing that Z is computationally indistinguishable from both X and Y. That is, we make the following two claims:

1. Z is computationally indistinguishable from Y:

   Note that the only difference between Z and Y is the witness that is used as input to the WI universal argument prover. Note also that the randomness used in running the WI prover is never referred to or used again in any other part of the simulation. Thus, this claim basically follows from the fact that WI is closed under concurrent composition. Still, as the protocol contains also messages that do not belong to the WI universal arguments, we prove the claim below.

   Let us order the sessions according to the scheduling of the first step in their second stage (the WI universal argument stage). Let Z_i denote a distribution where in the first i sessions we use follow the strategy for $\mathcal{S}im$ and in the last $n - i$ sessions we follow the strategy for $\mathcal{S}im'$. Note that $Z_0 = Z$ and $Z_n = Y$. It will be sufficient to prove that Z_i is computationally indistinguishable from Z_{i+1} for all $0 \leq i < n$.

   Indeed, suppose that we have a distinguisher D that distinguishes between Z_i and Z_{i+1} with probability ϵ. Let us fix a choice of coins used in all sessions before the $i + 1^{st}$ such that D distinguishes between Z_i and Z_{i+1} conditioned on this choice with probability ϵ. Note that the statement to be proved is now identical in Y and Z. Since the execution of all sessions other than the i^{th} is the same in Y and Z, and since the strategy for this execution is a function of the inputs, the coins, and possibly the public messages of the i^{th} session, the distinguisher D can be turned into a distinguisher for the WI universal argument system.

2. Z is computationally indistinguishable from X:

   The only difference between Z and X is that the commitment in Step P1.2 is for $h(\Pi)$ instead of $h(0^n)$. Note also that in both Z and X, the randomness used for this commitment is never referred to or used again in any other part of the simulation (because the WI stage uses y_i as a witness). Again, we prove the claim below, even though it is basically implied by the multiple-sample security of the commitment scheme.

   We now order the sessions according to the order of the message in Step P1.2. We define X_i to be a distribution where in the first i sessions we use $\mathsf{Com}(h(0^n))$ and in the last $n-i$ sessions we follow the strategy of $\mathcal{S}im$ (i.e., use $\mathsf{Com}(h(\Pi))$). Note that $X_0 = X$ and $X_n = Z$.

   Suppose that there exists a distinguisher D that distinguishes between X_i and X_{i+1} with probability ϵ. Suppose we fix a choice of coins for all sessions except the i^{th} such that D distinguishes between X_i and X_{i+1} conditioned on this choice with probability ϵ. The distinguisher D can be converted to a distinguisher between a commitment to $h(\Pi)$ and a commitment to $h(0^n)$ by hardwiring all messages before the message of Step P1.2 of the i^{th} session, and since the later messages are a function of the input and the previous messages.

□

We now mention some remarks regarding the proof and the protocol.

Remark 4.4.8. It is possible to define a "bounded-concurrent generation protocol" and to prove that Protocol 4.4.3 satisfies this definition. This, combined with the fact that WI is closed under concurrent composition, may be used to give more a modular proof of Theorem 4.4.6.

Remark 4.4.9. As observed by Yehuda Lindell, the proof of Theorem 4.4.6 actually yields a stronger statement than the theorem. For the purposes of the proof it does not matter if the message history consists of messages from an execution of our protocol, or from other executions of arbitrary protocol, as long as the history is short enough. Therefore, our protocol does not only compose concurrently with itself, but also with other protocols, as long as we have a bound on the total communication complexity of the other protocols. This property of our protocol was used in a recent work by Lindell [Lin03a].

Also note that for the proof it is not necessary for the messages themselves to be short. Rather, it is enough that they have a *short description*. To be more specific, for the proof to hold it is not necessary to have a the entire message history h be of length shorter than $n^4/2$. Rather, it is sufficient that there will be a polynomial-time computable function $F : \{0,1\}^{n^4/2} \to \{0,1\}^*$ to which the simulator can commit in advance such that there will exist an input x that satisfies $h = F(x)$. This input x will be the *short explanation* for h.

4.5 Obtaining a Zero-Knowledge Protocol Under Standard Assumptions.

In the previous sections, we have shown how to obtain non-black-box zero-knowledge arguments under the assumption of existence of hash functions that are collision-resistent against circuits of

Public input: 1^n: security parameter	1^n $\downarrow$ $\boxed{P}$ $\boxed{V}$
Step V1 (Choose random-access hash function): Verifier chooses a random index $\alpha \leftarrow_{\mathrm{R}} \{0,1\}^n$ for a random-access hash function ensemble $\{h_\alpha, \mathsf{cert}_\alpha\}_{\alpha \in \{0,1\}^*}$.	$\xleftarrow{\alpha \leftarrow_{\mathrm{R}} \{0,1\}^n}$
Step P2 (Commitment to hash of"junk"): Prover computes $z \leftarrow_{\mathrm{R}} \mathsf{Com}(h_\alpha(0^n))$ and sends z to verifier.	$\xrightarrow{z = \mathsf{Com}(h_\alpha(0^n))}$
Step V3 (Send random string): The verifier selects a string $r \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends it.	$\xleftarrow{r \leftarrow_{\mathrm{R}} \{0,1\}^n}$
The transcript of the protocol is the pair $\tau = (\alpha, z, r)$.	

Protocol 4.5.1. An alternative non-uniform verifier generation protocol

some "nice" super-polynomial bound (e.g., $n^{\log n}$). In this section, we show how we can relax the assumption and obtain the same results under the more standard assumption that there exist hash functions that are collision-resistent against polynomial-sized circuits. This section is based on the paper [BG01], which is joint work with Oded Goldreich.

Error-correcting codes. We use ECC to denote an *error correcting code*. That is, ECC is a polynomial-time computable function that satisfies the following:

Polynomial expansion: There exists some function $\ell(\cdot)$ such that $l(n) < n^c$ for some constant $c > 0$, and for every string $x \in \{0,1\}^*$, $|\mathsf{ECC}(x)| = \ell(|x|)$.

Constant distance: There exists some constant $\delta > 0$ such that for every $x \neq x' \in \{0,1\}^n$, $|\{i \mid \mathsf{ECC}(x)_i \neq \mathsf{ECC}(x')_i\}| \geq \delta \cdot \ell(n)$.

We note that we only need error correcting codes with *polynomial* expansion, even though there are known codes with *linear* expansion.[12]

Changing the definition of computational soundness. We make the following change in Definition 4.3.1. We relax the computational soundness requirement as follows: the original requirement required that the probability that a $T(n)^{O(1)}$-sized cheating prover outputs a witness is negligible. The relaxation we make is that we only require the soundness condition to hold against polynomial-sized provers. However, we require that such provers cannot even output an *implicit representation* of the witness (i.e., a circuit C such that $[C]$ is a witness). This is still a relaxation because in $T(n)^{O(1)}$ time it is possible to convert an implicit representation to an explicit representation. Because of the weak proof of knowledge condition of the universal arguments system, Theorem 4.3.2 still holds even if we use a generation protocol with this relaxed definition.

Our generation protocol. We now present a generation protocol that satisfies this relaxed definition – Protocol 4.5.1. It is very similar to Protocol 4.5.1, with a small difference.

[12] There are rather simple constructions for binary codes with polynomial expansion. For example, such codes can be obtained by concatenating the Reed-Solomon code with the Hadamard code (with the Hadamard code applied individually to each symbol of the Reed-Solomon code).

Definition of the language Λ. The language Λ is a slight modification over the language Λ defined in Section 4.3.3. We say that $\tau = (\alpha, z, r)$ is in Λ if there exists a circuit Π of size at most $T = |r|^{\log\log|r|/5c}$ such that $z = \mathsf{Com}(h_\alpha(\mathsf{ECC}(\Pi)))$ and $\Pi(z) = r$ (recall that $|\mathsf{ECC}(\Pi)| \leq |\Pi|^c$). A *witness* that $(h_\alpha, z, r) \in \Lambda$ is a triple (F, y, s) with the following properties:

- $z = \mathsf{Com}(y; s)$.
- F is a *certified* version of $\mathsf{ECC}(\Pi)$ with respect to the hash value y. This means, that if we let $\tilde{\Pi} = \mathsf{ECC}(\Pi)$ and $m = |\tilde{\Pi}|$, then F is a function on $[m]$ such that for every $i \in [m]$, $F(i) = (\tilde{\Pi}_i, \sigma)$, where σ is a certificate for $\tilde{\Pi}_i$ w.r.t. the random-access hashing scheme. More formally, F passes verification if for every $i \in [m]$, $V_{\alpha,y}(i, b, \sigma) = 1$ where $(b, \sigma) = F(i)$ and V is the verification algorithm of the random-access hashing scheme.
- Using the decoding algorithm for the error-correcting code it is possible to recover $\Pi = \mathsf{ECC}^{-1}(F)$. To be a valid witness, it must hold that $\Pi(z) = r$

It can be seen that this is indeed a valid witness relation for the language Λ and that $\Lambda \in$ **Ntime**$(n^{\log\log n})$. Thus, all that is left to prove is the following:

Claim 4.5.2. *Protocol 4.5.1 (w.r.t. the language Λ defined above) satisfies the relaxed definition of a generation protocol.*

Proof. Proving the simulation condition is almost identical to the case of Protocol 4.3.5, and so we will skip this part and prove only the computational soundness property. Suppose, towards a contradiction, that P^* is a cheating prover that contradicts our relaxed soundness condition. In the same way as the computational soundness proof of Section 4.3.4, we can use P^* to obtain in polynomial-time two certified encoding F and F' of two different programs Π, Π'. Now, if we choose i at random from $[m]$, then with constant probability $\mathsf{ECC}(\Pi)_i \neq \mathsf{ECC}(\Pi')_i$. This means that with constant probability $F(i)$ and $F'(i)$ yield two contradicting certificates for the same hash value, contradicting the security of the random-access hash value. $\square$

Is is straightforward to generalize this protocol to obtain a concurrent zero-knowledge protocol under standard assumptions. Indeed, the proof of the zero-knowledge property is virtually unchanged, while the proof of the soundness condition follows the proof of Claim 4.5.2. Note also that, as noted in Remark 4.3.3, the zero-knowledge protocol we obtain is not only sound but also a weak proof of knowledge (in the sense that a knowledge extractor can obtain a witness with probability that is polynomially related to the acceptance probability).

4.6 Conclusions and future directions

We have shown that (under standard complexity assumptions) there exists a protocol that can be shown to be zero-knowledge using a *non-black-box* simulator, but cannot be shown to be zero-knowledge using a *black-box* simulator. Moreover, we have shown that this protocol satisfies some desirable properties that are impossible to obtain when restricted to black-box simulation.

4.6.1 Reverse-engineering

Arguably, our simulator does not "reverse-engineer" the verifier's code, although it applies some non-trivial transformations (such as a **PCP** reduction and a Cook-Levin reduction) to this code.

Yet, we see that even without doing "true" reverse-engineering, one can achieve results that are impossible in a black-box model. This is in a similar vein to Chapter 3, where the impossibility of code obfuscation is shown without doing "true" reverse-engineering. One may hope to be able to define a model for an "enhanced black-box" simulator that would be strong enough to allow all the techniques we used, but weak enough to prove impossibility results that explain difficulties in constructing certain objects. We are not optimistic about such attempts.

4.6.2 Black-box impossibility results and concurrent zero-knowledge

There are several negative results regarding the power of *black-box* zero-knowledge arguments. The existence of non-black-box simulators suggests a re-examination of whether these negative results holds also for general (non-black-box) zero-knowledge. Indeed, we have already shown in this work that some of these results *do not* hold in the general setting. The case of concurrent composition is an important example. The results of [CKPR01] imply that (for a constant-round protocol) it is impossible to achieve even *bounded* concurrency using black-box simulation. We have shown that this result does not extend to the non-black-box settings. However, it is still unknown whether one can obtain a constant-round protocol that is (fully) concurrent zero-knowledge. This is an important open question.

4.6.3 Arguments vs. proofs

Our protocol is an *argument system* for **NP** and not a proof system because it is only computationally sound. However, we do not know whether there exists zero-knowledge *proof* for **NP** satisfying Properties 1-5 of Section 4.1.1. In particular, it is not known whether there exists a constant-round Arthur-Merlin (non-black-box) zero-knowledge *proof* system for **NP**.

4.6.4 Fiat-Shamir heuristic

The fact that we have shown a constant-round Arthur-Merlin zero-knowledge protocol, can be viewed as some negative evidence on the soundness of the Fiat-Shamir heuristic [FS86]. This heuristic converts a constant-round Arthur-Merlin identification scheme into a non-interactive signature scheme. The prover/sender in the non-interactive scheme uses the same strategy as in the interactive scheme, but the verifier/receiver's messages are computed by applying a public hash function to the message history, and so can be computed by the prover without any need for interaction.[13] It is known that the resulting signature scheme will be totally *insecure* if the original protocol is zero-knowledge [DNRS99]. Thus, the current work implies that there exist some constant-round Arthur-Merlin protocols on which the Fiat-Shamir heuristic cannot be applied. In a recent work, Goldwasser and Tauman [GT03] used our work to show a counter example for the Fiat-Shamir heuristic on 3 rounds.

An open question (related to the previous section) is whether or not the Fiat-Shamir heuristic is secure when applied to interactive *proofs* instead of arguments. In particular, an affirmative answer would imply that there does not exist a constant-round Arthur-Merlin zero-knowledge *proof* system for **NP**.

[13] This heuristic is usually applied to 3 round protocols, but it can be applied to any constant-round Arthur-Merlin protocol.

4.6.5 Number of rounds

Goldreich and Krawczyck [GK90] have shown that no 3 round protocol can be black-box zero-knowledge. We have presented several non-black-box zero-knowledge protocols, but all of them have more than 3 rounds. It is an open question whether there exists a 3-round zero-knowledge argument for a language outside **BPP**.

One bottleneck in reducing the number of rounds in our protocol is our use of a universal argument scheme, for which the best known construction utilizes 4 rounds. Thus, a related open question is whether or not there exist a non-interactive (or even two-round) universal argument scheme. Micali [Mic94] presents a candidate for such a scheme.

4.6.6 Strict polynomial-time

We have shown the first constant-round zero-knowledge protocol with a *strict* polynomial-time simulator, instead of an *expected* polynomial-time simulator. Another context in which *expected* polynomial-time arises is in constant-round zero-knowledge proofs *of knowledge* (e.g., [Fei90, Chap. 3], [Gol01b, Sec. 4.7.6.3]). In Chapter 5, we construct, using the protocol presented here, a constant-round zero-knowledge argument of knowledge with a *strict* polynomial-time extractor. We also show that non-black-box techniques are *essential* to obtain either a strict polynomial-time simulator or a strict polynomial-time knowledge-extractor, in a *constant-round* protocol.

Chapter 5

Applications of Non-Black-Box Zero Knowledge

Summary: We show three applications for the non-black-box zero-knowledge argument of Chapter 4. We prove that all these applications can *not* be obtained using black-box techniques.

The first application is a construction of a *resettably-sound* zero-knowledge argument. Loosely speaking, an argument system is *resettably-sound* if it remains sound even when the prover has the power of "rewinding" the verifier throughout their interaction. For example, this can be the case if the verifier is implemented in a "smart card" and the prover has the power to *reset* the card to its original factory settings. We also observe there does *not* exist a resettably-sound argument system for a non-trivial language that is zero-knowledge via a *black-box* simulator.

The second application is a construction of a *resettable zero-knowledge* argument of knowledge. A proof system is said to be *resettable zero-knowledge* if it remains zero knowledge even when the verifier has the power of "rewinding" the prover. It was observed previously that a resettable zero-knowledge argument of knowledge for a non-trivial can *not* have a *black-box* knowledge extractor.

The third application is a construction of a constant-round zero-knowledge argument for **NP** with a *strict* probabilistic-polynomial-time knowledge extractor. All previous such systems only possessed extractors that ran in *expected* probabilistic-polynomial-time. Again, we show that no such system can have a *black-box* knowledge extractor. In addition, we show that a constant-round zero-knowledge argument (for a non-trivial language) can not have a strict polynomial-time simulator.

5.1 Introduction

In this chapter, we apply the non-black-box zero knowledge argument of the previous chapter to obtain three new results. All these results are proven to be *impossible* to obtain using black-box techniques. Thus, the results of this chapter serve as an additional demonstration of the usefulness and importance of developing non-black-box techniques.

This chapter is based on the papers [BGGL01] and [BL02] which are joint works with Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell.

5.1.1 The Resettable Model

Two of the applications are for the *resettable* attack model introduced by Canetti, Goldreich, Goldwasser and Micali [CGGM00]. Recall that the strategy of a party P in an interactive protocol is specified by P's *next message function.* This is a function that, given the protocol's inputs, random tape, and list of messages that P received in the protocol up to a certain point, outputs the next message that P is going to send. When P participates in a single interaction with some party A, this means that A has very limited access to P's next-message function. This is because A can only obtain the value of the next message function on the messages that appeared in this single interaction. In particular, if A obtains the evaluation P's next-message function on the message list $\langle m_1, m_2 \rangle$, then it will not be able to obtain this evaluation also on a message list of the form $\langle m_1, m_2' \rangle$ where $m_2 \neq m_2'$.

In a *resettable* attack against a party P in an interactive protocol, we allow the adversary to have the power to query the next-message function of some party P on inputs of its choice. That is, we assume that the inputs to the protocol are fixed and that P's random tape is fixed, and allow the adversary to obtain P's next message function evaluated on any message list of her choice. This is a very strong form of attack and in particular it is stronger than previously considered attack models such as sequential, parallel, and concurrent composition.[1] The name "resettable" comes from the fact that after interacting with a party and sending it some messages $\langle m_1, m_2, \ldots, m_i \rangle$ the adversary has the power to "reset" the verifier to its initial condition and send the verifier a *different* list of messages.

The resettable model has both practical and theoretical motivations. On the practical side, this models a situation where a cryptographic protocol is implemented on a device such as a smart card, that doesn't have an internal source of fresh randomness. In such a setting, it is possible that when attacking the device an adversary will be able to reset the card back to its initial factory settings many times. On the theoretical side, it is very interesting whether one can obtain security against such strong attacks, especially since many of the known techniques for obtaining secure protocols (and in particular for obtaining zero knowledge proof systems) completely break down against a resettable attack. For more discussion on the resettable model, see the papers [CGGM00] and [BGGL01].

Ressetably-sound zero-knowledge. Our first result in this model is a construction of a zero-knowledge argument system for **NP** that is *resettably sound.* This means that this argument system preserves soundness even when a cheating prover can launch a "reset attack" against the honest verifier. We note that all previous constructions of zero knowledge systems were *insecure* against such an attack.[2] Indeed, intuitively, a black-box simulator is exactly an algorithm that uses access to the verifier's next-message function in order to prove possibly false statements to this verifier. (We will later see how to turn this intuition into a formal proof that there does not exist a resettably-sound zero-knowledge system with a black-box simulator for a non-trivial language.) Our proof is obtained by observing that any constant-round Arthur-Merlin proof or argument system, such as the zero-knowledge system constructed in the previous chapter, can be converted into a

[1] Loosely speaking, the difference between the resettable attack and all these attack modes (i.e., sequential , parallel, and concurrent composition) is that those attacks the adversary cannot query the next message function on incompatible transcripts (e.g., $\langle m_1, m_2 \rangle$ and $\langle m_1, m_2' \rangle$ where $m_2 \neq m_2'$) and the *same* random tape. Rather, in those attack models the honest party can use fresh random coins for each individual session, thus making it harder for the adversary to use what she learns in one session in order to attack a different session.

[2] We consider in this chapter only zero knowledge systems in the standard/plain model, without any setup assumptions such as existence of a third trusted party or a shared trusted reference string.

resettably-sound argument system in a simple way (by having the verifier use a pseudorandom function). This result, along with the observations above, can be seen as an alternative proof to the result of Goldreich and Krawczyk [GK90] that there does not exist a constant-round public-coin zero-knowledge argument or proof system for a non-trivial language that has a *black-box* simulator.

Ressetable zero-knowledge proof of knowledge. Our second result is a construction of an argument of knowledge for **NP** that is resettable zero-knowledge. That is, the argument system preserves the zero-knowledge property even when a cheating verifier can launch a "reset attack" against the honest prover. Our proof of knowledge has a *non-black-box* knowledge extractor. That is, the knowledge extractor uses the code of a cheating prover in order to extract the witness. Indeed, this is inherent as Canetti *et al.*[CGGM00] already observed that such a system cannot have a *black-box* knowledge extractor. This was considered to be a pity, since a natural application for the smart card setting are identification protocols, which often use zero-knowledge proofs of knowledge (e.g., the Fiat-Shamir paradigm [FS86]). Thus our construction enables a smart card to prove its identity using a proof of knowledge system that will remain zero-knowledge even under a reset attack. This construction combines ideas from [CGGM00] with the resettably-sound zero-knowledge system mentioned above.

5.1.2 Strict Polynomial-Time Knowledge Extraction

One of the properties of the zero-knowledge protocol of the previous chapter was that it is the first constant-round zero-knowledge protocol to have a *strict* probabilistic-polynomial-time simulator (i.e., the running time of its simulator is bounded by some polynomial in the security parameter). In contrast, previous constant-round zero-knowledge systems only had simulators that ran in *expected* polynomial-time. This means that if we consider the running time of the simulator as a random variable (on the space of the simulator's random tape) then the expectation of this random variable is bounded by some polynomial in the security parameter. However, it may be the case that with some probability the simulator runs in a *super-polynomial* number of steps.

Allowing the simulator to run in expected polynomial-time is somewhat undesirable for several reasons, both "philosophical" and technical. The philosophical reasons are that it is not clear that expected polynomial-time can be considered as an efficient computation, and also that it seems contrary to the spirit of the zero-knowledge definition to allow the simulator to run in expected polynomial-time, where the verifier must run in strict polynomial-time. The technical problems arise from the fact that is less understood than the more standard strict polynomial-time, which means that rigorous proofs of security of protocols that use zero-knowledge arguments with expected polynomial-time simulators as components, tend to be more complicated (c.f., [Lin01]).[3] In particular, expected polynomial-time simulation is not closed under composition, which can yield to problems when one needs to design a simulator for a large protocol that uses a zero-knowledge system as a sub-protocol. Also, the security reduction that is obtained by such protocols may be weaker than is sometimes desired. For example, assume that a protocol's security relies on a hard problem that would take 100 years to solve, using the best known algorithm. Then, we would like to prove that the probability that an adversary can successfully break the protocol is negligible, unless it runs for 100 years. However, when expected polynomial-time simulation is used, we cannot

[3]Because of these technical problems with expected polynomial-time, Levin ([Lev86], see also [Gol97]) suggested a different definition for expected polynomial-time that is more robust in some sense. However, even under Levin's definition, expected polynomial-time still has the mentioned drawbacks, and it still unclear whether it should be considered as efficient computation.

rule out an adversary who runs for 1 year and succeeds with probability 1/100. See [BL02] for a more thorough discussion on this matter.

Given the above discussion, a natural question (c.f., [Fei90, Sec. 3.2], [Gol01b, Sec. 4.12.3]) was whether one can obtain a constant-round zero-knowledge protocol with a strict polynomial-time simulator, and indeed this question is settled by the protocol of the previous chapter. However, there is still an analogous question for the case of *proofs of knowledge*. Recall that In a *proof of knowledge*, for every cheating prover P^* that manages to convince the honest verifier that some statement x is in some **NP**-language L with probability p, there exists a *knowledge extractor* that given access to this cheating prover P^*, will output a *witness* to the fact that $x \in L$ with probability p. In previous constructions of constant-round zero-knowledge proofs of knowledge, this knowledge extractor ran in *expected* polynomial-time.[4] Thus, the analogous question is whether there exists a constant-round zero-knowledge proof or argument system with a *strict* polynomial-time knowledge extractor.

In this chapter we solve this question in the affirmative. That is, we construct (using the protocol of the previous chapter) a constant-round zero-knowledge argument of knowledge for **NP** with a strict polynomial-time knowledge extractor. This knowledge extractor is *non-black-box* in the sense described above (i.e., it uses the code of the cheating prover to extract a witness). Indeed, we show that it is *impossible* to obtain both strict polynomial-time simulation and strict-polynomial extraction using black-box techniques.

5.1.3 Organization

Section 5.2 contains our construction for a resettably-sound zero-knowledge argument. Section 5.3 contains our construction for a resettable zero-knowledge argument of knowledge. This construction uses the construction of Section 5.2 and so Section 5.3 should be read after Section 5.2. Section 5.4 contains our construction for a zero-knowledge argument of knowledge with a strict polynomial-time knowledge extractor, along with the proof that strict polynomial-time knowledge extraction or simulation require non-black-box techniques. This section does not use results from the previous sections and so may be read independently of Sections 5.2 and 5.3.

Note. Throughout this chapter we focus on presenting these three applications, along with proof sketches. For more discussion, related results, and the full proofs, see the papers [BGGL01] and [BL02].

5.2 A Resettably-Sound Zero-Knowledge Argument

As mentioned above, a resettably-sound argument system is a system that preserves its soundness even when the prover has black-box access to the verifier's next-message functionality. The formal definition is a little more complicated because we strengthen the basic notion by allowing the adversary to choose several different statements on the fly.

---

[4] The way these previous knowledge extractors operated was roughly as follows: Let p be the probability that the cheating prover convinces the honest verifier that $x \in L$, then with probability approximately $1-p$ the extractor does almost nothing, and with probability p, the extractor runs for time $\frac{1}{p}q(n)$ (where $q(\cdot)$ is some polynomial) and outputs a witness. Note that the expected running time of such an extractor is less than $q(n)$. However $1/p$ may much larger than $q(n)$, and in this case if we halt this extractor after running $q(n)$ steps (or even $100q(n)$ steps), it will never output a witness (and thus it is not a strict polynomial-time extractor).

Definition 5.2.1. Let (P,V) be a argument system for some language L, we say that (P,V) is *resettably-sound* if for every polynomial $t(\cdot)$ and every polynomial-sized oracle machine A,

$$\Pr[A^{V(\cdot;r)}(1^n) = (x,\tau) \text{ s.t. } x \notin L \text{ and } \mathsf{Accept}_V(x,\tau;r) = 1] < \mathsf{neg}(n)$$

Where the oracle $V(\cdot;r)$ denotes the next message function of the verifier V with security parameter n, and a fixed random tape r (chosen at random from the set of strings of appropriate length). We say that $\mathsf{Accept}_V(x,\tau) = 1$ if τ is a transcript that the verifier V accepts as a proof that $x \in L$ when using r for a random tape.

We note that a natural and seemingly stronger definition will allow the adversary to interact with several copies of the verifier's next message function, where each copy uses an independent random tape. However, it can be shown that this seemingly stronger definition is in fact equivalent to Definition 5.2.1.

It is not hard to see that the standard examples of zero-knowledge systems, such as the proof system for 3-colorability of [GMW86], or the proof system for quadratic residuosity of [GMR85], are *not* resettably-sound. In fact, the standard proofs that these systems are zero-knowledge also imply the existence of a polynomial-time algorithm that can completely compromise the soundness of these systems under a reset attack on the verifier. Indeed, this is not a coincidence, as every black-box zero-knowledge system for a non-trivial language can *not* be resettably-sound. That is, we have the following theorem:

Theorem 5.2.2. *Suppose that (P,V) is a resettably-sound zero-knowledge argument system for a language L that has a black-box simulator. Then $L \in$ **BPP**.*

Proof Sketch: Let (P,V) be an argument system for L satisfying the conditions of the theorem, and let S be the black-box simulator for honest verifier of this system. For every $x \in L$, given black-box access to the verifier's next-message function, the simulator should output a transcript that is indistinguishable from a transcript of an interaction with the honest prover. In particular, this transcript will be an *accepting* transcript (i.e., a transcript that the honest verifier accepts as a proof that $x \in L$). However, if $x \notin L$, then the resettably-sound condition implies that the output of the simulator will *not* be an accepting transcript. Therefore, one can decide whether or not $x \in L$ by running the simulator and seeing whether or not it outputs an accepting transcript for x. □

Our main theorem of this section is the following:

Theorem 5.2.3. *Suppose that there exist collision-resistent hash functions. Then, there exists a resettably-sound zero-knowledge argument for* **NP**. *Furthermore, this argument system has a constant number of rounds.*

We prove Theorem 5.2.3 by showing a simple way to convert any constant-round public-coins argument system into a resettably-sound system. If the original argument system was zero-knowledge then so will be transformed system. We thus obtain Theorem 5.2.3 by combining this transformation with the constant-round public-coins zero-knowledge argument system of the previous chapter. We see that what we need is to prove the following lemma:

Lemma 5.2.4. *Let (P,V) be a constant-round public-coins argument system for a language L that satisfies a weak proof of knowledge property. Then, there is a proof system (P,V') for L that is resettably-sound.*

Notes: **(1)** Because only the verifier algorithm is changed in the new proof system, if the original proof system was zero-knowledge then so will be the transformed system **(2)** The weak proof of knowledge property is that if a prover P^* convinces the verifier to accept that a string x is in L with some non-negligible probability ϵ, then, given access to P^*, the knowledge extractor will obtain a witness to x with non-negligible probability ϵ'. We call it a *weak* proof of knowledge property because we don't require ϵ' to be negligibly close to ϵ. Note that the system of the previous chapter does indeed satisfy this property (c.f., Remark 4.3.3).

Proof Sketch: Instead of the original public-coin verifier V, the transformed verifier V' will use a *pseudorandom function* to compute its messages. That is, the random tape of V' will consist of a seed s for a pseudorandom function family $\{f_s\}_{s\in\{0,1\}^*}$, while the next-message function of V' will simply be the function f_s. That is, on input x, after receiving messages $m_1,\dots,m_i$, the verifier V' will respond with $f_s(x,m_1,\dots,m_i)$.

Now, in a reset attack of an adversary A against the verifier V', the oracle that the adversary gets access to is indistinguishable from a random function. We will use this to convert A into a cheating prover P^* for the original proof system (P,V) in the following way: Suppose that there are c verifier messages in the original proof system, and that the adversary A makes $q = q(n)$ queries to its oracle. We assume w.l.o.g that A never repeats the same query twice, also that if $x, \tau = \langle m_1,\dots,m_c\rangle$ is the transcript that A outputs at the end, then A queried its oracle all the prefixes $x, \langle m_1,\dots,m_i\rangle$ of this transcript in the course of the execution. (This is without loss of generality since A can be modified to behave in this order without affecting its output distribution and without incurring too much of a complexity overhead.)

The prover P^* for the system (P,V) will operate as follows: it will guess $l_1,\dots,l_c \in [q]$. Then it will simulate the execution A in the following way: it will answer all of A's queries with random strings, except for queries number $l_1,\dots,l_c$. The prover P^* will forward these queries to the verifier V, and return to A the response of V. We claim that if A had probability ϵ of success in its reset attack, then P^* will have probability $\frac{\epsilon}{q^c} - \mathsf{neg}(n)$ (this is the probability that P^* "guessed" the correct $l_1,\dots,l_c$ – note that this is polynomially related to ϵ since c is constant) of cheating the honest verifier V to accept x even though $x \notin L$.

This claim can be shown by first observing that if the verifier V' uses a *truly* random function then P^*'s simulation of A's execution is perfect, and hence in this case P^* will guess correctly the queries which correspond to A's final output transcript with probability at least $\frac{1}{q^c}$. We then note that if P^*'s probability of convincing the verifier V is non-negligibly smaller than $\frac{\epsilon}{q^c}$ then we can turn it into a distinguisher for the pseudorandom function ensemble. To make this distinguisher efficient one needs to use the knowledge extractor of the system (P,V) and this is where the weak proof of knowledge property is used. □

5.3 A Resettable Zero-Knowledge Argument of Knowledge

In this section we use the resettably-sound argument of Section 5.2 to construct a *resettable zero-knowledge argument of knowledge* for **NP**. The resettable zero-knowledge property guarantees that it is infeasible for an adversary to learn anything new about the statement proven even when given black-box access to the prover's next-message function. The argument of knowledge property guarantees that if one is given the actual *code* of some prover's next-message function, then it is in fact possible to learn "everything" about the statement proven. That is, one can use this code to efficiently extract a witness for the statement proven.

We start with the formal definition of resettable zero-knowledge.[5]

Definition 5.3.1. Let (P, V) be an argument system for some **NP** language L with witness relation R_L. We say that P is *resettable zero-knowledge* if there exists a probabilistic polynomial-time Turing machine S such that for every polynomial-sized oracle machine A and every $(x, y) \in R_L$ (where $x \in \{0,1\}^n$) the following two random variables are computationally indistinguishable:

- $A^{P(\cdot;x,y,r)}(1^n)$.
  $P(\cdot; x, y, r)$ denotes oracle access to the next-message function of P with fixed public input x, auxiliary input y and random tape r (where r is chosen uniformly among all strings of the appropriate length)

  and

- $S(A, x)$

We note that it is impossible to construct a non-trivial resettable zero-knowledge argument of knowledge with a *black-box* knowledge extractor.

Proposition 5.3.2. *Let (P, V) be a resettable zero-knowledge argument for a language L that is an argument of knowledge with a black-box knowledge extractor. Then $L \in$ **BPP**.*

Proof Sketch: Let E be the black-box knowledge extractor for (P, V). By the proof of knowledge property, for every $x \in L$, if we execute $E(x)$ with oracle access to the honest prover $P(\cdot; x, r)$ (where r is a random tape for P) then E will output a *witness* for x with probability very close to 1. On the other hand, such an execution is exactly a "reset attack" on P, and therefore if we run the simulator S on the description of E and x, we must still obtain a witness with probability close to 1. We see that to decide whether $x \in L$ or not, we can run $S(E, x)$: if $x \in L$ then we'll obtain a witness for x with probability close to 1. If $x \notin L$ then we'll certainly never obtain a witness for x (since no such witness exists). □

Our main theorem of this section is the following:

Theorem 5.3.3. *Suppose that there exist one-way-permutations secure and collision-resistent hash functions against 2^{n^ϵ}-sized circuits . Then, there exists a resettable zero-knowledge argument of knowledge for every language in* **NP**.

We note that the sub-exponential hardness assumption is not essential, and may be relaxed at the expense of a more complicated proof. See the paper [BGGL01] for more details.

5.3.1 Proof of Theorem 5.3.3

To prove Theorem 5.3.3 we use the approach of [CGGM00], which showed a way to transform (certain types of) *concurrent* zero-knowledge protocols into *resettable* zero-knowledge protocols. The transformation of [CGGM00] is not known to yield a proof/argument of knowledge, even if the original protocol had such a property. However, using the resettably-sound argument of Section 5.2,

[5] We note that our definition is slightly weaker (and simpler) than the definition of [CGGM00] since we allow the adversary to attack only one copy of the prover's next-message function, whereas [CGGM00] allowed the adversary access to several copies of the prover's next-message function (each with independent random tapes). Unlike in the case of soundness, in the case of zero-knowledge these two variants are not equivalent. However, our weaker and simpler definition does seem to capture the essence of resettable zero-knowledge and indeed all of our results generalize easily to the stronger definition. See [BGGL01] for more details.

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$)	$w \downarrow \boxed{P}$ $\quad x \downarrow \boxed{V}$
Step V0 (Verifier's commitment): Verifier sends commitments (using COM) to $x_1, \ldots, x_k \leftarrow_{\mathrm{R}} \{0,1\}^n$, where $k = k(n)$ is some unspecified function of n.	$\xleftarrow{\mathsf{COM}(x_1)\cdots\mathsf{COM}(x_k)}$
RK iterations: The following is performed k times for $i = 1, \ldots, k$: **Step Pi.1 (Prover's commitment):** Prover selects $\tilde{x}_i \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends $\mathsf{com}(\tilde{x}_i)$ to the verifier. **Step Vi.2 (Verifier's decommitment):** Verifier sends to prover the string x_i along with the decommitment information for x_i	$\xrightarrow{\mathsf{com}(\tilde{x}_i)}$ $\xleftarrow{x_i, \mathsf{d\text{-}info}(x_i)}$
Steps P,V$k+1 \ldots k+3$ (WI proof): Prover proves to verifier using the parallel-Hamiltonicity WI system that either $x \in L$ or that for some $i \in [k]$, one the prover's committed values $\tilde{x}_i$ was equal to x_i. The prover uses com as the commitment scheme for the Hamiltonicity proof system.	$w \downarrow \quad x, \tau \downarrow$ WIP $x \in L$ **or** $\exists_i \tilde{x}_i = x_i$ $\downarrow$ 0/1

Protocol 5.3.5. A concurrent zero-knowledge protocol

we will show a variant of this transformation that does preserve the proof of knowledge property. To keep the exposition simple, we will present a particular concurrent zero-knowledge argument and show how to transform this argument into a resettable zero-knowledge argument of knowledge (rather than showing a generic transformation, as is done in [CGGM00] and [BGGL01]).

The Concurrent Zero-Knowledge Protocol

We let com and COM to be two non-interactive perfectly binding and computationally hiding commitment schemes such that com can be completely broken in time $2^{n^{\epsilon/2}}$ (where n is the security parameter and $\epsilon > 0$ is some constant) but COM is secure even against $2^{n^{\epsilon}}$-sized circuits. Note that under our assumptions, such commitment schemes can be constructed using the scheme of [Blu82] with appropriately scaled security parameters. Our starting point is Protocol 5.3.5, which is a concurrent zero-knowledge argument for a language $L \in \mathbf{NP}$.

Protocol 5.3.5 is a variant of the concurrent zero-knowledge protocol of Richardson and Kilian [RK99], whose analysis was improved by Kilian and Petrank [KP00] and by Prabhakaran, Rosen and Sahai [PRS92]. We will use the following result of [PRS92]:

Theorem 5.3.4 ([PRS92]). *For every polynomial-time computable function k such that $k(n) = \omega(\log n)$ it holds that Protocol 5.3.5 which uses k iterations is a concurrent zero-knowledge argument.*

We remark that Protocol 5.3.5 is an argument of knowledge. This can be shown using the fact that the parallel-Hamiltonicity proof system is a proof of knowledge and using the fact that for every polynomial-sized prover P^* the probability that in an execution between P^* and the honest verifier it will be the case for some i that $\tilde{x}_i = x_i$ is negligible (as otherwise we can turn P^* into an $2^{o(n^\epsilon)}$-time algorithm that breaks the commitment scheme COM). Note that the knowledge extractor for Protocol 5.3.5 is a *black-box* knowledge extractor.

The Resettable Zero-Knowledge Protocol.

Protocol 5.3.5 is certainly not a resettable zero-knowledge protocol as it is a proof of knowledge with a black-box knowledge extractor. Intuitively, the problem seems to be the verifier's query in the WI proof system phase. The verifier can run the same interaction up to the WI system phase and then use two different queries in this phase to obtain the witness for the statement proven. One way to ensure that the verifier cannot perform this attack is to force the verifier to commit in the beginning of the protocol to the query that it is going to use in the WI stage. Indeed, this can be done to obtain a resettable zero-knowledge protocol (if the prover uses as randomness a pseudorandom function applied to the verifier's first message). Unfortunately, it seems that by doing this modification we lose the proof of knowledge property of the original protocol.

To obtain our resettable zero-knowledge argument of knowledge we use a variant of this modification. The verifier does indeed commit to its query q at the beginning of the protocol. However, the verifier never reveals the decommitment information for q in the course of the protocol. Instead, after sending q as its query in the WI proof, the verifier proves that this is the same string it committed to initially using the resettably-sound zero-knowledge argument of Section 5.2. Another modification is that the prover's random tape will be a seed for a pseudorandom function, and it will apply this function to the verifier's first message to obtain the random choices for the rest of the protocol. The modified protocol is Protocol 5.3.6. We shall now sketch the proof that Protocol 5.3.6 is a resettable zero-knowledge argument of knowledge.

Lemma 5.3.7. *Protocol 5.3.6 is an argument of knowledge (with a non-black-box knowledge extractor).*

Proof Sketch: Let P^* be some (possibly cheating) prover that manages to convince the honest verifier that $x \in L$ with some non-negligible probability p. By the properties of the parallel-Hamiltonicity protocol, to extract a witness for x it is enough for the verifier to obtain two responses γ, γ' to two different queries $q \neq q'$ for the same prover initial message α.

The knowledge extractor obtains the first such response by simply executing internally a normal interaction between the honest verifier and the prover P^*. After simulating this interaction the knowledge extractor rewinds the prover P^* to the point just after the prover sent the first message α of the WI proof system. The extractor then sends a new random query q', uses the *simulator* to prove the (false) statement that q' is consistent with initial commitment. If the extractor obtains a response then it can compute a witness. Otherwise it continues until such a response is obtained.[6] The zero-knowledge property and the computational hiding property of the commitment scheme ensure that the prover will not be able to distinguish between the "fake" and "real" executions, and so the extractor will have the same success probability as the prover's probability of convincing the verifier in a real interaction. □

Lemma 5.3.8. *Protocol 5.3.6 is resettable zero-knowledge.*

[6]Actually, one needs to introduce a timeout mechanism to ensure that it runs in expected polynomial-time. We note that the formal analysis is somewhat easier for an extractor that uses the simulator and a random (inconsistent) query also in the initial execution.

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$)	$\overset{w}{\downarrow}\ \overset{x}{\downarrow}$ $\boxed{P}\quad\boxed{V}$
Modification: Prover's random tape is $s \leftarrow_{\mathrm{R}} \{0,1\}^n$ - a random seed for a pseudorandom function ensemble $\{f_s\}_{s\in\{0,1\}^*}$ and $u \leftarrow_{\mathrm{R}} \{0,1\}^n$ – a random tape for the verifier algorithm of the resettably-sound zero-knowledge argument of Section 5.2	
Step V0 (Verifier's commitment): Verifier sends commitments (using COM) to $x_1,\ldots,x_k \leftarrow_{\mathrm{R}} \{0,1\}^n$, where $k = k(n)$ is some unspecified function of n. **Modification: (1)** In addition the verifier commits to a random string $q \leftarrow_{\mathrm{R}} \{0,1\}^n$ (which will serve as the verifier's query in the WI proof stage). **(2)** The prover uses for the rest of the protocol as a random tape for the prover of Protocol 5.3.5 the value $r = f_s(m)$ where m is the verifier's message.	$\xleftarrow{\mathsf{COM}(x_1)\cdots\mathsf{COM}(x_k),\mathsf{COM}(q)}$
RK iterations: The following is performed k times for $i = 1,\ldots,k$: **Step Pi.1 (Prover's commitment):** Prover selects $\tilde{x}_i \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends $\mathsf{com}(\tilde{x}_i)$ to the verifier. **Step Vi.2 (Verifier's decommitment):** Verifier sends to prover the string x_i along with the decommitment information for x_i	$\xrightarrow{\mathsf{com}(\tilde{x}_i)}$ $\xleftarrow{x_i,\mathsf{d\text{-}info}(x_i)}$
Steps P,V$k+1\ldots k+O(1)$ (WI proof): Prover proves to verifier using the parallel-Hamiltonicity WI system that either $x \in L$ or that for some $i \in [k]$, one the prover's committed values $\tilde{x}_i$ was equal to x_i. The prover uses com as the commitment scheme for the Hamiltonicity proof system. **Modification:** The verifier uses the string q as its query in this system. However, the verifier does *not* send the deccommitment information for q but rather proves that q is consistent with the initial commitment using a *resettably* sound zero-knowledge argument. The prover uses u as the random tape for the verifier in this proof. In the schematic description we denote by α and γ the prover's first and last messages of the WI proof system.	$\xrightarrow{\alpha}$ $\xleftarrow{q}$ r-sound ZKA q consistent $\xrightarrow{\gamma}$

Protocol 5.3.6. A resettable zero-knowledge argument of knowledge

Proof Sketch: We note some properties of Protocol 5.3.6 that will be useful for us in the analysis.

- Note that the honest prover algorithm never uses its private random tape when deciding whether or not to abort the execution. Also the decision whether to abort or not does not depend on previous messages of the prover but rather it can be computed efficiently from the list of verifier messages sent so far in the execution.
- We note that the particular commitment scheme COM that is used (i.e., Blum's scheme [Blu82]) has the property that not only the plain-text message but also the decommitment information is uniquely determined by the commitment string.

Let V^* be a cheating verifier that launches a reset attack against the prover of Protocol 5.3.6. We will show how can transform the verifier V^* into a verifier $\tilde{V}$ that launches a *concurrent* attack on the concurrent zero-knowledge Protocol 5.3.5 such that the outputs of the two verifiers are indistinguishable. Thus, we can use the concurrent simulator for Protocol 5.3.5 to simulate the resetting verifier V^*.

The verifier $\tilde{V}$ run an internal copy of V^* and will operate as follows: (We will sometimes refer to $\tilde{V}$ as the *external* verifier)

- Recall that the internal verifier V^* is an oracle machine that makes queries of the form $\langle m_1, \ldots, m_i \rangle$ (for $i \leq l$) to its oracle, where the oracle's response should be the prover's response in a session where the first i verifier messages are $m_1, \ldots, m_i$. We will make some assumptions on the internal verifier V^*'s behavior. These assumptions are without loss of generality since we can always modify V^*'s behavior (without changing its output distribution) so that it satisfies them.

  1. We assume without loss of generality that the verifier V^* does not ask its oracle a query for which the answer is obviously "abort". Because the prover's choice to abort can always be publicly computed from the transcript, we can assume that the internal verifier V^* never asks a query for which the answer is "abort".
  2. We assume without loss of generality that the internal verifier V^* never asks the same query twice.
  3. We assume that the internal verifier V^* never asks a query $\langle m_1, \ldots, m_i \rangle$ of its oracle without asking before it all prefixes of this query (i.e. all queries of the form $\langle m_1, \ldots, m_j \rangle$, where $j < i$).

- We are now ready to describe the operation of the external verifier $\tilde{V}$:

  1. The verifier $\tilde{V}$ uses a random $u \leftarrow_{\mathrm{R}} \{0,1\}^n$ for a random tape.
  2. If the internal verifier V^* sends a sequence containing only the initial message $m_1 = \mathsf{COM}(x_1), \ldots, \mathsf{COM}(x_k), \mathsf{COM}(q)$ the external verifier starts a new concurrent session with the prover and sends the prover the message $\mathsf{COM}(x_1), ldots, \mathsf{COM}(x_k)$. It forwards the prover's response to the internal verifier. We call the message m_1 (which uniquely identifies the session) the *key* of this session.
  3. If the internal verifier V^* sends a sequence $\langle m_1, \ldots, m_i \rangle$, then under our assumptions it has previously sent the sequence $\langle m_1, \ldots, m_{i-1} \rangle$ in the session keyed by m_1. The external verifier's behavior differs according to the step in the protocol m_i belongs to:

(a) If m_i is a message of the form $x_i, \mathsf{d\text{-}info}(x_i)$ (where by non-abort assumption this message is determined by the commitment in message m_1) then the external verifier sends m_i to the prover in the session keyed by m_1 and forwards the prover's response to the internal verifier.

(b) If m_i is the query q of the WI system stage, then the external verifier does *not* forward q to the prover. Rather, it answers the query itself, by using the honest verifier algorithm for the resettably-sound zero-knowledge argument with random tape u. This is what the external verifier does also if m_i is any intermediate message of the resettably-sound zero-knowledge argument (i.e., it does not forward anything to the external prover but answers using the honest verifier algorithm for the resettably-sound protocol).

(c) If m_i is the last message of the resettably-sound argument (and by our non-abort assumption the verification of the argument passed successfully) then the external verifier send the query q contained in the sequence $\langle m_1, \ldots, m_i \rangle$ to the prover in the session keyed by m_1 and forwards the prover's response to the internal verifier. Note that by the resettable soundness of the zero-knowledge argument system we can assume that in this case q is indeed the value committed to in the message m_1 (since this will be false only with negligible probability).

4. Note that we will not get "stuck" and try to send to the prover two different i^{th} messages in the same session, since all messages we forward are completely determined by the first message m_1.

- At the end of the execution, the external verifier $\tilde{V}$ outputs whatever the internal V^* does.

We claim that the output of $\tilde{V}$ in a concurrent execution is indistinguishable from the output of V^* in a reset attack. Indeed, we can see that $\tilde{V}$ always responds to a query its internal V^* makes with the honest prover's next-message function applied to this query. The only difference is that in the concurrent setting, in each concurrent session the honest prover uses a fresh and independent random tape. However, because each concurrent session is keyed by the verifier's initial message, this is exactly the same as if the prover used as a random tape the result of a random function applied to the verifier's initial message. This is computationally indistinguishable from the execution of V^* in a reset attack, where the prover uses a pseudorandom function. □

5.4 Strict Polynomial-Time Simulation and Extraction

The zero-knowledge argument system constructed in Chapter 4 was the first constant-round system to have a *strict* probabilistic polynomial-time knowledge extractor. In this section, we use that scheme to construct a constant-round zero-knowledge argument of knowledge that has a strict probabilistic polynomial-time *knowledge extractor*. We also show that both these applications are impossible to obtain using black-box techniques. To summarize, this section contains the following theorems:

Theorem 5.4.1. *Suppose that there exist trapdoor permutations over $\{0,1\}^n$ and collision-resistant hash functions. Then, there exists a* constant-round *zero-knowledge argument of knowledge for* **NP** *with a* strict *polynomial-time knowledge extractor and a* strict *polynomial-time simulator.*

Theorem 5.4.2. *There do not exist* constant-round *zero-knowledge proofs or arguments with* strict polynomial-time black-box *simulators for any language $L \notin$* **BPP**.

Theorem 5.4.3. *There do not exist* constant-round *zero-knowledge proofs or arguments of knowledge with* strict polynomial-time black-box *knowledge extractors for any language* $L \notin \mathbf{BPP}$.

In this section we provide merely the main ideas behind the proofs of these theorems. The full proofs can be found in the paper [BL02].

5.4.1 Proof Sketch of Theorem 5.4.1 – Construction of a Zero-Knowledge Argument of Knowledge with Strict Polynomial-Time Extraction

Recall the definition of a *commit-with-extract scheme* (c.f. Section 2.5). The following proposition allows us to reduce proving Theorem 5.4.1 to the problem of constructing a constant-round commit-with-extract scheme with a strict polynomial-time commitment extractor.

Proposition 5.4.4. *Suppose that there exists a constant-round commit-with-extract scheme with a strict polynomial-time extractor and a constant-round zero-knowledge argument system for* **NP**. *Then, there exists a constant-round zero-knowledge argument of knowledge for* **NP** *with a strict polynomial-time knowledge extractor.*

Indeed Proposition 5.4.4 reduces the problem of constructing a zero-knowledge argument of knowledge with a strict polynomial-time extractor to the problem of constructing:

1. A constant-round zero-knowledge argument system for proving *membership* (not knowledge) in **NP** language. For this we can use the argument system of Chapter 4.[7]

2. A constant-round commit-with-extract scheme with a *strict* probabilistic polynomial-time commitment extractor.

Proof sketch of Proposition 5.4.4. Assuming we have these two components, our construction for a zero-knowledge argument of knowledge is depicted in Protocol 5.4.5. Basically, the protocol works by the prover first committing to a witness using the commit-with-extract scheme, and then using the zero-knowledge proof of membership to prove that the committed string is indeed a witness.

Loosely speaking , the simulator for this protocol will work by committing to a "junk" string (e.g., the all-zeroes string) in the first step, and then using the simulator for the membership proof in the second step. The extractor for this scheme will work by running the extractor for the commit-with-extract scheme. The soundness of the zero-knowledge membership proof system of the second phase will ensure that the extracted string is indeed a witness.

□

Construction of a Commit-With-Extract Scheme.

The "heart" of the proof of Theorem 5.4.1 is the construction of a constant-round commit-with-extract scheme with a strict probabilistic polynomial-time knowledge extractor. This construction is depicted in Protocol 5.4.6.[9]

[7] Alternatively, if we are willing to allow the *simulator* of the resulting zero-knowledge argument of knowledge to run in expected polynomial-time (while still requiring that the knowledge extractor runs in strict polynomial-time), then we can also use any other constant-round zero-knowledge argument for **NP**, such as the argument system of Feige and Shamir [FS89].

[8] We note that we assume that the commit-with-extract scheme satisfies the *public decommitment* property, and so this is indeed a valid **NP** statement.

[9] For simplicity, we present the construction of a *bit* commitment scheme, the extension to n-bit strings can be done by committing to each individual bit in parallel, assuming that the zero-knowledge system used is n-time parallel-zero-knowledge (which is a weaker condition than bounded concurrent zero-knowledge).

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$)	$\overset{w}{\downarrow}\ \boxed{P} \qquad \overset{x}{\downarrow}\ \boxed{V}$
Steps P,V1.x (Commit to witness): Prover commits to the witness w using a *commit-with-extract* scheme. (Actually, the commit-with-extract scheme is an *interactive* protocol, and so this step will take a (constant) number of communication rounds.)	$\xrightarrow{z = \mathsf{Comm\text{-}Ext}(w)}$
Steps P,V2.x (ZK membership proof): Prover proves to verifier using a ZK membership proof that the string committed to in the previous step is a witness for x.[8]	$\boxed{ZKP\ \mathsf{Comm\text{-}Ext}^{-1}(z) \in R(x)}$

Protocol 5.4.5. A zero knowledge argument of knowledge with strict polynomial-time extraction.

We now sketch why Protocol 5.4.6 is indeed a commit-with-extract scheme.

Perfect binding. Since we assume that the permutation ensemble is certified, we may assume that the function f is a permutation (as otherwise the receiver would abort). Given that, and for any particular execution transcript containing f, r_1, r_2, τ, there exist only one r and σ such that $f(r) = r_1 \oplus r_2$ and $h(r) \oplus \sigma = \tau$.

Computational hiding. Intuitively, because r_2 is chosen at random, the string $r_1 \oplus r_2$ is uniform, and so distinguishing between a commitment to 0 and a commitment to 1 entails distinguishing between $f(U_n), h(U_n)$ and $f(U_n), h(U_n) \oplus 1$. Thus one can reduce the hiding property of this commitment scheme to the hiding property of Blum's commitment scheme (i.e., $\mathsf{Com}(\sigma; r) = f(r), h(r) \oplus \sigma$) [Blu82]. This intuition is indeed behind the proof that this commitment scheme satisfies the binding property, but there are some complications in the proof, see [BL02] for more details.

Commitment extractor. Recall that a commitment extractor is given a (possibly cheating) sender algorithm S^* and needs to output a simulated interaction between the sender S^* and the honest receiver, along with the value of the bit that was committed in this interaction. The extractor for the commitment works in the following way. It executes internally an interaction with the sender plays the part of the receiver, but instead of sending a commitment to r_1 in Step R2, it sends a commitment to a "junk" string (e.g., the all-zeros string). Then, after seeing r_2, it chooses r at random and lets $r_1 = f(r) \oplus r_2$. It then sends to S^* the string r_1, and uses the *simulator* of the zero-knowledge system to simulate a proof of the (false) statement that r_1 is the string it committed to in Step R2. Once S^* sends τ, the extractor computes its auxiliary output $\sigma = \tau \oplus h(r)$. Note

[10] Recall that a permutation ensemble is *certified*, if it is possible for the verifier to make sure that the function f is indeed a permutation. We can also use a non-certified trapdoor permutation ensemble, by having the sender prove in zero-knowledge that f was chosen from ensemble after Step S1.

Public input: 1^n (security parameter) **Sender's auxiliary input:** $\sigma \in \{0,1\}$ (bit to be committed to)	$\overset{w}{\downarrow}$ $\overset{x,r}{\downarrow}$ $\boxed{S}$ $\boxed{R}$
Choose trapdoor permutation **Step S1 (Choose trapdoor function):** Sender chooses a trapdoor permutation f, along with its associated trapdoor f^{-1} and sends f to the receiver. We assume that f is *certified*.[10]	$\xrightarrow{f}$
Coin-tossing **Step R2 (Choose r_1):** Receiver chooses $r_1 \leftarrow_{\mathrm{R}} \{0,1\}^n$, and sends a commitment to r_1, using a standard (perfect binding, computationally hiding) commitment scheme Com. That is, it chooses random coins s for the commitment scheme and sends $z = \mathsf{Com}(r_1; s)$ to the sender. **Step S3 (Choose r_2):** Sender chooses $r_2 \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends r_2 to receiver. **Step R4 (Reveal r_1):** Receiver sends r_1 to the sender but *without* sending also the decommitment information s. Rather, it proves in zero-knowledge that the sent value is indeed the value committed to in Step R2. The zero-knowledge system used in this phase has a *strict* polynomial-time *simulator*.	$\xleftarrow{z = \mathsf{Com}(r_1; s)}$ $\xrightarrow{r_2}$ $\xleftarrow{r_1}$ $\overset{z, r_1}{\downarrow}$ $\overset{s}{\downarrow}$ $\boxed{\begin{array}{l} ZKP \\ r_1 = \mathsf{Com}^{-1}(z) \end{array}}$ $\downarrow$ $0/1$
Commitment **Step S5 (Commitment):** Sender computes $r = f^{-1}(r_1 \oplus r_2)$ and sends $\tau = h(r) \oplus \sigma$ to the receiver, where $h(\cdot)$ is a *hard-core predicate* of f.	$\xrightarrow{\tau = h(f^{-1}(r_1 \oplus r_2)) \oplus \sigma}$

To decommit, sender sends r and σ, the receiver verifies that $f(r) = r_1 \oplus r_2$ and $h(r) \oplus \sigma = \tau$.

Protocol 5.4.6. A Commit-With-Extract Scheme that for this extractor to run in strict polynomial-time it is clear that the zero-knowledge system used in Step R4 has a strict polynomial-time simulation.

Combined with Proposition 5.4.4, the construction of this commit-with-extract scheme completes the proof of Theorem 5.4.1. □

5.4.2 Proof Ideas for Theorems 5.4.2 and 5.4.3 – Black-Box Lower Bounds.

In this section, we show the ideas behind the proofs of Theorems 5.4.2 and 5.4.3. That is, we show that there does not exist a constant-round zero-knowledge argument (resp., argument of knowledge), with a black-box simulator (resp., extractor) that runs in strict polynomial-time.

Before presenting the proofs, we motivate why it is not possible to obtain strict polynomial-time black-box extraction for constant-round protocols. First, consider a very simple (cheating) prover P^* who at every step either aborts or sends the honest prover message. Furthermore, the probability that it does not abort at any given step is $\epsilon = \epsilon(n)$. Then, black-box extractors for constant-round protocols would typically extract a witness using the following strategy: Invoke an execution with P^* and if P^* aborts, then also abort. However, if P^* does not abort (and thus sends a prover message in some crucial round), then continually rewind P^* until *another* prover message is obtained for this round. Obtaining two (or possibly more) different prover messages then suffices for extracting the witness. Now, this second case (where P^* does not abort) occurs with probability only ϵ. However, this means that the expected number of rewinding attempts by the extractor equals $1/\epsilon$. Therefore, the overall expected amount of work is bounded. However, since ϵ can be any non-negligible function, we cannot provide *any* strict polynomial upper-bound on the running time of the simulator.

This idea underlies our lower bounds. Specifically, we show that by carefully choosing the abort probabilities, it is possible to achieve the following effect: A strict polynomial-time black-box extractor will not have "time" to obtain two non-abort responses from the prover P^* in any given round. (By "not having time", we mean that with noticeable probability, the extractor will have to wait longer than the bound on its running-time in order to see a second non-abort response.) Essentially, this means that the extractor cannot "rewind" the prover. This suffices for proving the lower bound because, informally speaking, black-box extractors must be able to rewind in order to extract. The same argument also holds for strict polynomial-time simulation of constant-round zero-knowledge proofs.

Outline of the proofs. As we have mentioned above, the underlying idea behind the proofs of both Theorem 5.4.2 and Theorem 5.4.3 is the same. We will explain the intuition behind this idea in the context of knowledge extraction (i.e., in the context of the proof of Theorem 5.4.3) and then describe how this intuition generalizes also to the context of simulation (for the proof of Theorem 5.4.2).

Theorem 5.4.3 is proven by showing that if a language L has a constant-round zero-knowledge protocol (P, V) with a black-box strict polynomial-time knowledge extractor, then there exists a cheating verifier strategy V^*, such that for every $x \in L$, if V^* interacts with the honest prover P, then with noticeable probability, V^* will obtain a witness w for x from the interaction with P. Of course, the ability to do this contradicts the zero-knowledge property of the protocol, unless the verifier V^* could anyway obtain a witness by itself. Since V^* runs in probabilistic polynomial-time, it must therefore be the case that $L \in \mathbf{BPP}$ (because that's the only way that V^* could obtain a witness by itself).

We construct this verifier V^* from the black-box knowledge extractor of the system (P, V). Loosely speaking, this is done in the following way:

1. We let $x \in L$ and consider a cheating prover strategy P^* which is of the following form: P^* behaves exactly as the honest prover P behaves on input x, except that in each round of the proof, it may choose to abort the execution with some probability. We will choose these probabilities so that P^* will still have a noticeable probability to convince the honest verifier to accept x.[11]

2. Consider an execution of the knowledge extractor when it is given black-box access to P^*. Every query that the extractor makes to the black-box is a list of messages $(\alpha_1, \ldots, \alpha_i)$, and the reply received by the extractor is what P^* would send in a real execution when the first i verifier messages were $\alpha_1, \ldots, \alpha_i$. Thus, if P^* would abort in the real execution then the extractor receiver $\perp$. Note that at the end of the execution the knowledge extractor should output a witness for x with some noticeable probability.

   We show that it is possible to choose P^*'s abort probabilities in such a way, that if we run the knowledge extractor with black-box access to P^* then with very high probability, the extractor will get at most c non-abort replies, where c is the number of verifier messages in the protocol.

   Furthermore, these replies will correspond to i queries (where $i \leq c$) of the form (α_1), (α_1, α_2), …, $(\alpha_1, \ldots, \alpha_i)$ where $\alpha_1, \ldots, \alpha_i$ are some strings. That is, all these queries are prefixes of a single sequence $(\alpha_1, \ldots, \alpha_i)$.

3. We then implement a verifier strategy V^* that sends these messages $\alpha_1, \ldots, \alpha_i$ when it interacts with the prescribed prover P. Basically, the verifier works by internally running the knowledge extractor. When the extractor makes a query, the verifier either answers it with $\perp$ or forwards the query to the prover, and then returns the prover's reply to the knowledge extractor. We show that the verifier has a noticeable probability of perfectly simulating P^* to the knowledge extractor. In the case that this happens, the verifier will be able to obtain a witness for x with noticeable probability, which is what we wanted to prove.

Choosing the abort probabilities. We'll use the following choice for the prover P^*'s abort probability. After receiving the i^{th} verifier message, the prover P^* will continue (i.e., not abort) with probability $\epsilon^{2^{c-i}}$. Where $\epsilon = \epsilon(n)$ will be chosen to be an inverse polynomial smaller than $\frac{1}{t(n)^3}$, where t is the number of queries the knowledge extractor makes to its oracle. The decision to continue or abort would be based on a random tape obtained by applying a t-wise independent hash function to the current message history, for each query of the knowledge extractor, the decision of P^* whether to answer this query with $\perp$ (i.e., abort) will be independent. One can see that in a real interaction, the prover P^* will convince the verifier with probability

$$p^* = \epsilon^{2^{c-1}} \epsilon^{2^{c-2}} \cdots \epsilon^{2^0} = \epsilon^{\sum_{j=0}^{c-1} 2^j} = \epsilon^{2^c - 1}$$

[11] We will use a strategy for P^* that causes it to abort with quite high probabilities. In particular, P^* will abort in each round with probability greater than $1/2$ (and even greater than $1 - 1/n$). Note however that since the number of rounds in the protocol is constant, this does not preclude P^* from causing the honest verifier to accept with noticeable probability.

On the other hand, the probability that the extractor gets two non-$\perp$ replies to two distinct queries that are *not* prefixes of one another is significantly lower. That is, the probability that the extractor gets two non-$\perp$ replies to queries of the form $\langle \alpha_1, \ldots, \alpha_{i-1}, \alpha_i \rangle$ and $\langle \alpha_1, \ldots, \alpha_{i-1}, \alpha'_i \rangle$, where $\alpha_i \neq \alpha'_i$ is at most

$$\epsilon^{2^{c-1}} \epsilon^{2^{c-2}} \cdots \epsilon^{2^{c-i}} \epsilon^{2^{c-i}} = \epsilon^{2^c}$$

which is a factor ϵ less than p^*. Even if we take a union bound over all the t^2 possible query pairs of the extractor, this is still noticeably less than p^*. This means that there is a noticeable probability that the extractor succeeds in obtaining a witness in an execution where it did *not* obtain non-$\perp$ replies to two queries that are not prefix of one another.

As mentioned above, in the simulation case (i.e., when proving Theorem 5.4.2) we use a similar technique. Basically, we prove Theorem 5.4.2 by showing that if L has a constant-round zero-knowledge proof or argument system (P, V) with a black-box strict polynomial-time simulator, then there exists a cheating prover strategy P^* that does not have any auxiliary input (like a witness), and yet for every $x \in L$ causes the honest verifier V to accept x with noticeable probability. We stress that, unlike the honest prover, this cheating strategy P^* does *not* get a witness for x as auxiliary input. It is not hard to show that the existence of such a strategy P^* implies that $L \in \mathbf{BPP}$.

In summary, both impossibility results are due to the following two facts:

1. Intuitively, in order to successfully extract or simulate, the extractor/simulator must see at least two different continuations of the same transcript. That is, it must get meaningful replies to queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \alpha_{i-1}, \alpha'_i)$ where $\alpha'_i \neq \alpha_i$. Otherwise the extractor/simulator does not have any advantage over the interactive verifier/prover. (Informally speaking, "rewinding" is essential for black-box simulation and extraction.)

2. For any strict polynomial-time extractor or simulator, there exist provers/verifiers for which the time needed to obtain meaningful (i.e., non-abort) replies to two different continuations is beyond the scope of the extractor's/simulator's running-time.

Chapter 6

Non-Malleable Cryptography

Summary: We construct the first *constant-round* non-malleable commitment scheme and the first *constant-round* non-malleable zero-knowledge argument system, as defined by Dolev, Dwork and Naor. Previous constructions either used a non-constant number of rounds, or were only secure under stronger setup assumptions. An example of such an assumption is the *shared random string model* where we assume all parties have access to a reference string that was chosen uniformly at random by a trusted dealer.

We obtain these results by defining an adequate notion of *non-malleable coin-tossing*, and presenting a *constant-round* protocol that satisfies it. This protocol allows us to transform protocols that are non-malleable in (a modified notion of) the *shared random string model* into protocols that are non-malleable in the *plain* model (without any trusted dealer or setup assumptions). Observing that known constructions of a non-interactive non-malleable zero-knowledge argument systems in the shared random string model are in fact non-malleable in the modified model, and combining them with our coin-tossing protocol we obtain the results mentioned above.

The techniques we use are different from those used in previous constructions of non-malleable protocols. In particular our protocol uses diagonalization and a *non-black-box* proof of security (in a sense similar to the zero-knowledge argument of Chapter 4).

6.1 Introduction

6.1.1 Overview

In the *man-in-the-middle* (MIM) attack on a cryptographic two-party protocol, the adversary has complete control over the communication channel between two honest parties. The adversary has the power not only to read all messages sent between the parties, but also actively change, erase or insert its own messages into the channel. This attack can be very relevant in many practical settings, and thus designing protocol secure against such an attack is an important task.

Dolev, Dwork and Naor [DDN91] considered the MIM attack and defined a protocol to be *non-malleable* if it remains secure under such attack.[1] In the *plain* model (where there are no setup assumptions such as a public key infrastructure or a shared random string), [DDN91] constructed non-malleable protocols for the fundamental cryptographic tasks of commitment and zero-knowledge.

This chapter is based on the paper [Bar02].

[1]The definition of non-malleability is usually described in somewhat different terms. See also Remark 6.1.1

However, the protocols of [DDN91] took a *non-constant* number of communication rounds (logarithmic in the security parameter). In this work, we focus on the task of constructing *constant-round* protocols for these tasks.

The shared random string model. Unlike the case in the plain model, in the *shared random string model*, introduced by Blum, Feldman and Micali [BFM88], one assumes that there exists a trusted party that chooses a string uniformly at random and sends it to all parties (including the adversary) before the protocol is executed. This setup assumption enables the construction of much more round efficient protocols. In fact, Sahai [Sah99] constructed a 1-round (i.e., *non-interactive*) non-malleable zero-knowledge argument for **NP** in this model, whereas Di Crescenzo, Ishai and Ostrovsky [DIO98] constructed a 1-round non-malleable commitment scheme in this model (see Section 6.1.3 for details on other constructions and related works in this model).

Our results and techniques. In this paper we give the first construction of *constant-round* non-malleable commitment and zero-knowledge schemes in the *plain* model, without any setup assumption. Our approach is to first construct a non-malleable *coin-tossing* protocol. We then use this coin-tossing protocol in order to transform a non-malleable protocol (such as a zero-knowledge proof or a commitment scheme) from the shared random string model into the plain model.

The techniques utilized in the construction of the non-malleable coin-tossing protocol are different from those used in previous works in non-malleable cryptography. In particular our proof of security involves a diagonalization argument and a non-black-box use of the code of the adversary's algorithm. Similar techniques were first used in [Bar01] in the context of zero-knowledge systems. This works demonstrates that these techniques are applicable also in other settings in cryptography.

6.1.2 Model and Basic Terminology

In this work we are only interested in two-party protocols. We will denote the two parties by the letters L and R (L stands for left, R for right). Two examples that are worth keeping in mind are *commitment schemes* and *zero-knowledge proofs*. In a *commitment scheme* the left player L is the *sender* that wants to commit to a value, while the right player R is the *receiver* that receives the committed value. In a *zero-knowledge proof* the left player L is the *prover* that wants to convince the right player R (called the *verifier*) that some statement is true.

In the *man-in-the-middle setting* (MIM), as depicted in Figure 6.1, there is a third party denoted by C (C can stand for either *center* or *channel*, we will typically call C the *adversary*). All the communication between L and R is done through C. Thus, both players L and R only talk to C and cannot communicate directly with each other. The adversary C can decide to simply relay the messages each party sends to the other party, but it can also decide to block, delay, or change messages arbitrarily. Thus, if L and R wish to run a two-party protocol Π in the MIM setting, then we can think of the protocol Π as being executed in *two concurrent* sessions. In one session L plays the left side and the adversary C plays the right side, and in the second session C plays the left side and R plays the right side. We assume that the adversary C controls the scheduling of messages in both sessions. We call the first session (where L interacts with C) the *left* session, and the second session (where C interacts with R) the *right* session.

Unavoidable strategies.

There are two strategies that the adversary C can always use without being detected. One strategy is the *relaying* strategy in which the only thing C does is relay the messages between L and R. In

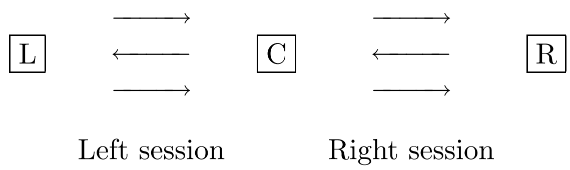


Figure 6.1: The MIM setting: all communication between L and R is done through the adversary C.

this case C is transparent and this is equivalent to a single execution of the protocol Π between L and R. The other unavoidable strategy is the *blocking* strategy in which C plays its part in each session completely independent of the other session. In each session, C uses the honest strategy to play its part (i.e., in the left session C uses the strategy of the honest right player and in the right session C uses the strategy of the honest left player.) Clearly, regardless of the protocol Π, it is impossible to prevent the adversary from using one of these two strategies. Therefore, we call the relaying and blocking strategies the *unavoidable* strategies. Intuitively, the goal in designing protocols for the man-in-the-middle setting, is to design protocols that force C to use one of the two unavoidable strategies (or such that it could not be advantageous to C to use any other strategy).[2]

For example, consider the case of a commitment scheme. When executed in the man-in-the-middle setting, in the left session the player L commits to a value α to C that plays the receiver, whereas in the right session C plays the sender and commits to a value $\widetilde{\alpha}$.[3] If C uses the *relaying* strategy then it holds that $\alpha = \widetilde{\alpha}$. On the other hand, if C uses the *blocking* strategy then it holds that $\widetilde{\alpha}$ is independent of α. Indeed, loosely speaking, the goal in *non-malleable* commitments is to design a commitment scheme such that regardless of the strategy C uses, it will hold that either $\widetilde{\alpha}$ is equal to α or that $\widetilde{\alpha}$ is independent of α.[4]

Remark 6.1.1 (Comparison with [DDN91]). Dolev *et al.*[DDN91] used different notations to describe essentially the same setting. They considered four parties P_1, P_2, P_3, P_4 that execute a two-party protocol in two concurrent sessions (see Figure 6.2). The left session is between P_1 and P_2, and the right session is between P_3 and P_4. Both P_2 and P_3 are controlled by an adversary, whereas P_1 and P_4 are honest and follow the protocol (and thus, P_1 is oblivious to the (P_3, P_4) interaction and P_4 is oblivious to the (P_1, P_2) interaction). This means that P_2 and P_3 combined correspond to the adversary C in our notation, and that P_1 corresponds to L in our notation, where P_4 corresponds to R in our notation. Previous works also used somewhat different emphasis in presenting the goal of non-malleable protocols. For example, the goal of a non-malleable commitment scheme is usually described as to ensure that the committed values in both sessions are *independent* (i.e., that the adversary is using the *blocking* strategy). The possibility of the values being *identical* (i.e., that the adversary will use the *relaying* strategy) is also allowed because it is unavoidable, but it is considered to be an uninteresting special case. In contrast, we treat both strategies equally. Note also that in this work we only consider protocols that are non-malleable with respect to *themselves* (as defined by [DDN91]), where [DDN91] defined also non-malleability with respect to general protocols.

[2] Actually, the adversary can always use also a "mixed" strategy in which it follows the relaying strategy with probability p, and the blocking strategy with probability $1 - p$, for some $p \in [0, 1]$.

[3] We only consider *statistically binding* commitment schemes, and so the committed value is determined uniquely by the transcript of the session.

[4] Actually, one needs to define an appropriate notion of "computational independence", as is done in [DDN91]. See also Section 6.4.

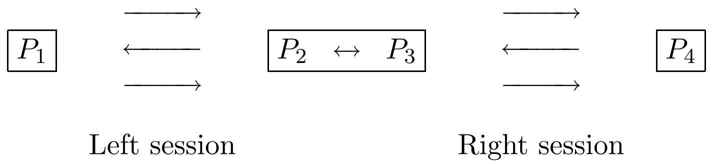


Figure 6.2: Non-malleability, as defined by [DDN91] : P_2 and P_3 are controlled by the adversary

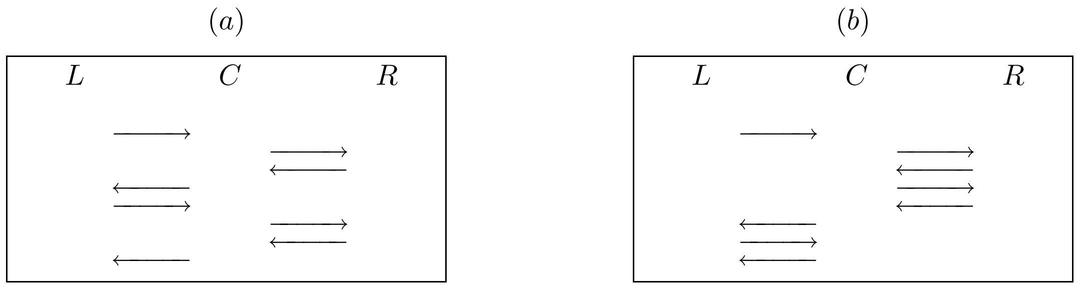


Figure 6.3: Two different scheduling strategies: (a) The man in the middle applies the "synchronizing" strategy. (b) The man in the middle does not wait for L's reply before answering R's message.

Scheduling strategies

The adversary in the MIM model can control not only *what* is written in the messages sent but also decide *when* to send them. That is, the adversary has complete control over the scheduling of the messages. An important example of a scheduling strategy is the *synchronizing* scheduling. In this scheduling, the adversary synchronizes the two executions by immediately sending the i^{th} message in the right session after it receives the i^{th} message in the left session and vice versa (i.e., it sends the j^{th} message in the left session immediately after it received the j^{th} message in the right session). We call an adversary that always uses this scheduling a *synchronizing* adversary. We call an adversary that uses a different scheduling a *non-synchronizing* adversary. The difference between a synchronizing and a non-synchronizing adversary is illustrated in Figure 6.3. Note that even when restricted to the synchronizing scheduling, it is still possible for the adversary to use either the blocking or the relaying strategies. Thus, these strategies remain unavoidable even when restricting to synchronizing adversaries.

Unlike other settings in cryptography, such as concurrent zero-knowledge [DNS98, RK99, CKPR01, PRS92], in our setting the ability to control the scheduling does *not* add much power to the adversary. In fact, as we will show in Section 6.5, it is possible to transform any non-malleable commitment or zero-knowledge scheme that is secure against *synchronizing* adversaries, into a scheme secure against *general* (possibly non-synchronizing) adversary. Therefore, in most of this paper we will restrict ourselves into dealing only with synchronizing adversaries.[5] As mentioned above, in Section 6.5 we will show how to overcome this restriction.

[5]In some sense, the synchronizing scheduling is the hardest schedule to deal with when designing a non-malleable protocol. In fact, one intuition behind the non-malleable commitment protocol of [DDN91] (and also an earlier work in a different model by Chor and Rabin [CR87]) is that the protocol is designed to force the adversary to use a *non-synchronizing* strategy.

6.1.3 The Shared Random String Model

In the *shared random string model* [BFM88], we assume the existence of a third trusted party called the *dealer*. Before a protocol is executed in this model, the dealer picks a string r uniformly at random and sends it to all the parties. The string r is called the *reference string*, and is given as an additional public input to the protocol. In the setting we are interested in (the MIM setting for a two-party protocol), this means that in a preliminary phase, the dealer sends the string r to L, R and C. After this phase, the protocol is executed in both the left and right sessions, with r as the public reference string.

Previous works. As mentioned above, unlike the case in the plain model, there are several round-efficient non-malleable protocols known in the shared random string model. Sahai [Sah99] constructed a single-round (i.e., *non-interactive*) non-malleable zero-knowledge proof system in this model. This scheme was improved by De Santis, Di Crescenzo, Ostrovski, Persiano and Sahai [DDO$^+$01]. Di Crescenzo, Ishai and Ostrovsky [DIO98] constructed in the shared random string model a non-interactive commitment scheme that is non-malleable in a weaker sense than [DDN91] ("non-malleable w.r.t. opening" [FF00]). Di Crescenzo, Katz, Ostrovski, and Smith [DKOS01, Sec. 3] constructed in the shared random string model[6] a non-interactive commitment satisfying the stronger notion of non-malleability (i.e., "non-malleable w.r.t. committing") defined in [DDN91]. Canetti and Fischlin [CF01] constructed in the shared random string model[7] non-interactive *universally composable* commitments which is a stronger notion than non-malleability. Interestingly, it is *impossible* to construct universally composable commitments in the plain model [CF01].

6.1.4 Non-malleable Coin-Tossing

The goal of this paper is to convert two-party protocols that are secure against a MIM attack in the *shared random string* model, into protocols that are secure against such an attack in the *plain* model. Toward this end we will want to construct a *non-malleable coin-tossing protocol* (in the plain model). Once we have such a coin-tossing protocol, we will convert a two-party protocol Π_{Ref} with a reference string that is secure in the shared random string model into a protocol Π_{Plain} in the plain model in the following way:

Construction 6.1.2 (Composition of a coin-tossing protocol with a reference string protocol).

Phase 1: Run the coin-tossing protocol. Let r denote the result of this execution.

Phase 2: Run the protocol Π_{Ref} using r as the common reference string.

Loosely speaking, our goal is to construct a coin-tossing protocol such that whenever Π_{Ref} was secure in the shared random string model, the protocol Π_{Plain} will be secure in the plain man-in-the-middle model (with no shared string).

[6] Their construction is in the common reference string (CRS) model which is a slight generalization of the shared random string model. However they remark that under standard assumptions (e.g., hardness of factoring), their construction can be implemented in the shared random string model.

[7] Footnote 6 (Yehuda Lindell, personal communication, April 2002).

The modified shared random string model

Suppose that we execute the protocol Π_{Plain}, as constructed in Construction 6.1.2, in the man-in-the-middle setting. Let r denote the result of Phase 1 (the coin-tossing protocol) in the left session and let $\tilde{r}$ denote the result of Phase 1 in the right session. If we wish to emulate exactly the shared random string model then we want to ensure that $r = \tilde{r}$. Indeed, this will be the case if C will act transparently (i.e., use the *relaying* strategy, as described in Section 6.1.2). However, we have no guarantee that C will indeed use this strategy. In fact, C can always ensure that $\tilde{r}$ is independent of r, by using the *blocking* strategy.

The above problem motivates us in defining (for the MIM setting) a new ideal model called the *modified shared random string model*. In this model, the trusted dealer generates *two* random strings $r^{(1)}$ and $r^{(2)}$ uniformly and independently of one another. Then $r^{(1)}$ is used in the left session (between L and C), but the adversary C is allowed to choose whether it wants to use the same string $r^{(1)}$ also in the right session (between C and R), or whether it wants to use the new independent string $r^{(2)}$. We allow the adversary to view the two strings before it makes this decision.[8]

Intuitively, it seems that the ability to choose that the strings used in both sessions will be independent should not help the adversary. Rather, it will only make the information that the adversary receives in the left session useless in the right session, and vice versa. Indeed, it turns out that many known protocols that are secure in the shared random string model, are also secure in the *modified* shared random string model. Thus we set our goal to constructing a coin-tossing protocol that would allow us to convert any protocol Π_{Ref} secure in the *modified* shared random string model into a protocol Π_{Plain} secure in the plain MIM setting. We provide an adequate definition, which we call *non-malleable coin-tossing protocol*, that indeed achieves this goal.

Definition of non-malleable coin-tossing

A *non-malleable coin-tossing* protocol is a protocol that implements the ideal functionality of the modified shared random string model, in the real (or plain) model, where there is no trusted third party. The formal definition is as follows:

Definition 6.1.3 (Non-malleable coin-tossing). Let $\Pi = (L, R)$ be a (plain) two-party protocol. We say that Π is a non-malleable coin-tossing protocol if the following holds. For any efficient algorithm C there exists an efficient algorithm $\widehat{C}$ such that the following random variables are computationally indistinguishable:

1. $\mathsf{output}_{(L,R,C),\Pi}(1^n)$ where this denotes the triplet of outputs of L,R and C when executing Π in two concurrent sessions.
2. $(r^{(1)}, r^{(b)}, \tau)$ where this triplet is generated by the following experiment: first $r^{(1)}, r^{(2)}$ are chosen uniformly and independently in $\{0,1\}^n$. Then we let $(b, \tau) \leftarrow \widehat{C}(r^{(1)}, r^{(2)})$.

Definition 6.1.3 follows the paradigm of simulating an adversary C in the real model (i.e., the plain MIM setting) by an ideal adversary $\widehat{C}$ in the ideal model (i.e., the modified shared random string model). Indeed, Item 1 corresponds to the output of all parties in when the coin-tossing protocol is executed in the plain MIM model with adversary C, while Item 2 is the output of all parties when they interact with the trusted dealer of the modified shared random string model with adversary $\widehat{C}$.

[8] We do not know whether or not it is unavoidable to allow the adversary to view both strings or at least one of them, if we want a model that can be simulated in the plain MIM setting.

6.1.5 Our Results

Our main result is the following:

Theorem 6.1.4. *Suppose that there exist hash functions that are collision-resistent against 2^{n^ϵ}-sized circuits for some $\epsilon > 0$. Then, there exists a constant-round non-malleable coin-tossing protocol.*

By following Construction 6.1.2, and composing the coin-tossing protocol of Theorem 6.1.4 with a non-malleable protocol in the shared random string model, such as the non-interactive zero-knowledge of [DDO$^+$01],[9] we obtain the following theorem:

Theorem 6.1.5. *Suppose that there exist trapdoor permutations and collision-resistent hash functions strong against 2^{n^ϵ}-sized circuits for some $\epsilon > 0$. Then:*

1. *There exists a constant-round non-malleable zero-knowledge argument system for* **NP**.
2. *There exists a constant-round non-malleable (statistically binding) commitment scheme.*

Remarks. We note that our result can be obtained also somewhat weaker complexity assumptions. Note also that the non-malleable commitment scheme we obtain is non-malleable with respect to committing non-malleable *with respect to committing* (as the definition of [DDN91], which stronger than the definition of [DIO98], see [FF00]). We also note that, like the previous protocols of [DDN91], our schemes are *liberal* non-malleable in the sense that our simulator runs in *expected* polynomial-time. However, we believe that one can obtain the stronger notions using the techniques of Barak and Lindell [BL02]. See Section 6.6 for more discussion and details.

General theorems. It would have been nice if we proved two general statements of the form (**1**) "every protocol that is secure in the shared random string model is also secure in the *modified* shared random string model" and (**2**) "for every protocol that is secure in the modified shared random string model, its composition using Construction 6.1.2 with a non-malleable coin-tossing protocol yields a protocol that is secure in plain MIM setting". However this is problematic, not so much because Definition 6.1.3 is too weak, but mainly because the notion of "security" for protocols is not well-defined and depends on the particular application. We do however prove more general statements than Theorem 6.1.5: see Section 6.4 for more details.

6.1.6 Organization

In Section 6.2 we construct a non-malleable coin-tossing protocol that is secure against *uniform* polynomial-time adversaries. In Section 6.3 we show how to modify the construction to obtain security against *non-uniform* adversaries. In Section 6.4 we show how we can use our non-malleable coin-tossing protocol to obtain a non-malleable zero-knowledge and commitment schemes. In Section 6.5 we show how we can convert a non-malleable zero-knowledge or commitment scheme that is secure against adversaries that use the *synchronizing* scheduling, into a scheme that is secure general adversaries, that may use different scheduling strategies. Section 6.6 contains some remarks on the constructions and open questions.

[9] Actually, we will use a variation of this protocol, which will be interactive (but constant-round), in order to avoid assuming the existence of dense cryptosystems.

We note that if one wants just to "get a taste" of our techniques and constructions, it is possible to read just Sections 6.2.1 and 6.2.2 to see a simple construction of a simulation sound zero-knowledge system secure against uniform adversaries. Simulation soundness is an important relaxation of non-malleability, and this construction illustrates some of the ideas used in the other sections.

6.1.7 Cryptographic assumptions.

For the purposes of this chapter, we will assume the existence of collision-resistant hash functions that are secure against circuits of sub-exponential size (i.e. $2^{n^{\epsilon}}$ for some fixed $\epsilon > 0$).[10] Using an appropriate setting of the security parameter we will assume that all cryptographic primitives we use are secure against 2^{n^5}-sized circuits, where n is the security parameter for our protocol.[11] In contrast we aim to prove that our protocol is secure only against adversaries that use *uniform* probabilistic polynomial-time algorithms.[12]

6.2 A Uniform Non-Malleable Coin-Tossing Protocol

In this section we will construct a non-malleable coin-tossing protocol. The protocol of this section has two limitation: The first limitation is that our protocol will only secure against adversaries that use *uniform* probabilistic polynomial-time algorithms (rather than polynomial-sized circuits or equivalently, polynomial-time algorithms with auxiliary input). The second limitation is that our protocol will only be secure against adversaries that use the *synchronizing* scheduling. As mentioned in Section 6.1.5, constructing a non-malleable coin-tossing protocol against synchronizing adversaries is sufficient for our desired applications (since in Section 6.5 we show how to transform a non-malleable zero-knowledge or commitment scheme secure against synchronizing adversaries into a scheme secure against general adversaries.)

Therefore it is the first limitation (security only against uniform adversaries) that is actually more serious. Note that usually in cryptography, it *is* possible to derive security against non-uniform adversaries from a security proof against uniform adversaries. This is because most proofs of security use only *black-box* reductions. However, this is *not* the case here, since we will use some diagonalization arguments in our proof of security that do not carry over to the non-uniform case. Nonetheless, in Section 6.3 we do construct a different non-malleable coin-tossing protocol that is secure against *non-uniform* adversaries.

Rough outline of proof structure. The general form of our non-malleable coin-tossing protocol is similar to previous (malleable) coin-tossing protocols. In fact, it is quite similar to a coin-tossing protocol of Lindell [Lin01], except for the following modification: The modification is that while the protocol of [Lin01] involves a zero-knowledge proof that some condition X occurs, in our protocol we prove that *either* X or Y occurs, where Y is some "bogus" condition that almost always will *not*

[10] As mentioned in Section 6.1.5, our protocol can be proven secure under somewhat weaker assumptions at the cost of a more complicated analysis, see Section 6.6.

[11] For example, if we have a primitive that is secure against $2^{m^{\epsilon}}$ sized circuit with security parameter m, then when given the security parameter n as input, we will invoke the primitive with $m = n^{5/\epsilon}$.

[12] The main limitation of the current protocol is that it is only secure against *uniform* adversaries rather than the quantitative difference (2^{n^5} vs. polynomial-time) between the hardness assumption and the adversary's running time. Indeed, our protocol is in fact secure against uniform $2^{n^{\delta}}$-time algorithms for some $\delta > 0$. However, for the sake of clarity, we chose to model the adversary as a uniform probabilistic *polynomial-time* algorithm. The protocol of Section 6.3 is also in fact secure against $2^{n^{\delta}}$-sized circuits for some $\delta > 0$.

be satisfied in a real execution. This is a technique that originated in the work of Feige, Lapidot and Shamir [FLS99], and has been used in several places since (see also Chapter 4). When this technique is used it is usually the case that one can ensure that Condition Y occurs if one has the power to "rewind" the adversary. Thus, it is usually the case that the *adversary simulator* ensures that Condition Y occurs in the simulation.[13] This will *not* be the case here. Although we do need to provide an adversary simulator (or equivalently, an ideal adversary) $\widehat{C}$ to satisfy Definition 6.1.3, our simulator will *not* use the bogus Condition Y and in fact Condition Y will not occur even in the simulation. In fact, Condition Y will be of a form that no polynomial-time algorithm will be able to ensure it occurs, even with the use of rewinding. If we're not using this condition, then what do we need it for? The answer is that we will use this condition in the security proof. In order to show that our actual simulator $\widehat{C}$ does satisfy the conditions of Definition 6.1.3 we will construct an "imaginary simulator" $\widehat{C}''$. This "imaginary simulator" will run in time that is super-polynomial and will use this long running time instead of rewinding to ensure that Condition Y occurs.[14] Using the output of this "imaginary simulator" as an intermediate hybrid we will be able to prove that our actual simulator satisfies the conditions of Definition 6.1.3.

6.2.1 Evasive Sets

We will need to use the existence of an (exponential-time constructible) set $R \subseteq \{0,1\}^*$ that is both pseudorandom and hard to hit in the following sense:[15]

Definition 6.2.1 (Evasive set). Let $R \subseteq \{0,1\}^*$. For any $n \in \mathbb{N}$ denote $R_n \stackrel{def}{=} R \cap \{0,1\}^n$. We say that R is *evasive* if the following conditions hold with respect to some negligible function $\mu(\cdot)$:

Constructibility: For any $n \in \mathbb{N}$, the set R_n can be constructed in time 2^{n^3}. That is, there exists a 2^{n^3} time Turing machine M_R that on input 1^n outputs a list of all the elements in the set R_n. In particular this means that deciding whether or not $r \in R$ can be done in time $2^{|r|^3}$.

Pseudorandomness: The set R is pseudorandom against uniform probabilistic polynomial-time algorithms. That is, for all probabilistic polynomial-time Turing machines M, it holds that

$$\left| \Pr_{r \leftarrow_{\mathrm{R}} R_n}[M(r)=1] - \Pr_{r \leftarrow_{\mathrm{R}} \{0,1\}^n}[M(r)=1] \right| < \mathsf{neg}(n)$$

.

Evasiveness: It is hard for probabilistic polynomial-time algorithms to find an element in R_n. Furthermore, even when given an element $r \in R_n$, it is hard for such algorithms to find a (different) element in R_n. Formally, for any probabilistic polynomial-time Turing machine M and for any $r \in R_n$,

$$\Pr[M(r) \in R_n \setminus \{r\}] < \mathsf{neg}(n)$$

Sets with very similar properties have been shown to exist by Goldreich and Krawczyk [GK92]. Using similar methods we can prove

Theorem 6.2.2. *Suppose that 2^{n^ϵ}-strong one-way-functions exist, then there exists an evasive set.*

[13] This is the case also in Chapter 4, although there the simulator used the knowledge of the adversary's code instead of rewinding to ensure that Condition Y occurs.

[14] In the non-uniform version of our protocol, the "imaginary simulator" will need to use also the knowledge of the adversary's code (in addition to a longer running time) to ensure that this condition occurs. See Section 6.3.

[15] For simplicity we "hardwired" into Definition 6.2.1 the constants and time-bounds required for our application.

Proof. First note that when constructing the set R_n it is enough to ensure that R_n satisfies the pseudorandomness and evasiveness properties only for probabilistic Turing machines whose description is of size at most $\log n$ and whose running-time[16] is at most $n^{\log n}$. This will ensure that the set $R = \cup_{n\in\mathbb{N}} R_n$ will be pesudorandom and evasive for all probabilistic polynomial-time Turing machines. We denote the set of Turing machines that have size at most $\log n$ and running time halted at $n^{\log n}$ by $\mathcal{M}_n$. Note that $|\mathcal{M}_n| \leq n$.

We fix $f : \mathbb{N} \to \mathbb{N}$ be a (polynomial-time computable) function such that $f(\cdot)$ is super-polynomial and $f(n) = 2^{o(n^\epsilon)}$. For concreteness, we will set $f(n) = n^{\log n}$. Suppose that we choose at random a subset $S = \{x_1, \ldots, x_f\}$ of size $f(n)$ (That is, $x_1, \ldots, x_f$ are chosen uniformly and independently in $\{0,1\}^n$). Let $\mu(n) \stackrel{def}{=} f(n)^{-1/3}$. Note that $\mu(\cdot)$ is a negligible function. By the chernoff inequality, for every Turing machine M, the probability (over the choice of S) that $|\mathbb{E}_{x\leftarrow_{\mathrm{R}}\{0,1\}^n}[M(x)] - \mathbb{E}_{x\leftarrow_{\mathrm{R}}S}[M(x)]| > \mu(n)$ is extremely low ($2^{-\Omega(f(n)^{1/3})}$). Thus with high probability S is pseudorandom for all the machines $M \in \mathcal{M}_n$.

Let $i \neq j \in [f]$ and let $M \in \mathcal{M}_n$. The probability over S that $\Pr[M(x_i) = x_j] > \mu(n)$ is at most $\frac{2^{-n}}{\mu(n)}$ (where $\mu(\cdot)$, as before is defined by $\mu(n) \stackrel{def}{=} f(n)^{-1/3}$). By taking a union bound over all possible such pairs (i, j) we see that with the overwhelming probability of at least $1 - \frac{2^{-n}f(n)^2}{\mu(n)}$, the set S will be evasive for all the machines $M \in \mathcal{M}_n$.

Under our assumptions, there exists a generator $G : \{0,1\}^{\mathrm{polylog}(n)} \to \{0,1\}^{n^{\log n}}$ such that $G(U_n)$ is pseudorandom for circuits of size at most $n^{\log^2 n}$. That is, for any circuit with input $m = m(n)$ and size at most $2^{n^{\epsilon'}}$,

$$\| \Pr[C(U_m) = 1] - \Pr[C(G(U_n)) = 1] \| < n^{-\log^2 n}$$

.

Given a set $S \subseteq \{0,1\}^n$ of size $f(n)$ we can use this generator to verify in deterministic $2^{\mathrm{polylog}(n)}$-time that it is indeed satisfies the pseudorandomness and evasiveness properties for the machines in $\mathcal{M}_n$. Indeed, this can be done by using the generator to decrease the number of coin tosses of all Turing machines in $\mathcal{M}_n$ to $\mathrm{polylog}(n)$ and then simulating all machines in $\mathcal{M}_n$ in time $2^{\mathrm{polylog}(n)}$ (since enumerating all possible $\mathrm{polylog}(n)$ coin tosses for each machine can be done in time $2^{\mathrm{polylog}(n)}$). This means that we have a deterministic $2^{\log^c(n)}$-time test T (for some constant $c > 0$) such that given a set $S = \{x_1, \ldots, x_f\}$, if T outputs 1 then S is evasive, and $\Pr[T(U_{n\cdot f(n)}) = 1] > 1 - \mu'(n)$ for some negligible function $\mu'(\cdot)$.

Under our assumptions, there exists also a pseudorandom generator $G' : \{0,1\}^{\mathrm{polylog}(n)} \to \{0,1\}^{2^{\log^c(n)}}$. We use this generator G' to *find* a set S that satisfies these properties in time $2^{\mathrm{polylog}(n)}$ (which is at most $2^{O(n)}$) in the following way: go over all possible seeds and stop at the lexicographically first seed s such that $T(G(s)) = 1$.

Note that for our purposes it is enough to use a pseudorandom generator that runs in time that is *exponential* in its seed length. Such generators exist under weaker conditions than the ones stated in the theorem [NW88, IW97]. □

6.2.2 A Simple Simulation-Sound Proof System

In this section, we sketch the construction and proof of a simple simulation sound proof system that is based on the notion of evasive sets. Loosely speaking, a proof system is *simulation sound*

[16] We can assume that all Turing machines are "clocked". That is, they are of the form that first computes 1^t and then runs for at most t steps.

Public input: 1^n: security parameter, $x \in \{0,1\}^{n^\epsilon}$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$)	$w \downarrow$ $\boxed{P}$ $x \downarrow$ $\boxed{V}$
Step P1 (Send r): Prover sends to verifier a random string r of length $n - n^\epsilon$ **Steps P,V2.x (ZK Proof):** Prover proves to verifier using its input w via a zero knowledge universal argument that either $x \in L$ or that $x \circ r \in R$ where R is the evasive set. Verifier accepts if proof is completed successfully.	$\xrightarrow{r \leftarrow \{0,1\}^{n-n^\epsilon}}$ $w \downarrow$ $x, r \downarrow$ $\boxed{\begin{array}{l} ZK\text{-}proof \\ x \in L \textbf{ or} \\ x \circ r \in R \end{array}}$ $\downarrow$ 0/1

Protocol 6.2.3. A Simulation-Sound Zero-Knowledge Protocol

[Sah99] if even when simulating a man-in-the-middle adversary, we still are ensured (with very high probability) that if the adversary's proof in the simulated right session passes verification, then either statement is a copy of the statement proven in the left session, or the statement is true. This should hold even if the statement proven in the simulated left session is false. Note that if we use a standard (i.e., standalone) zero-knowledge proof system in the man-in-the-middle setting, it may be that when we simulate the left session, the adversary manages to prove a false statement in the right session. Simulation soundness is a weaker condition than non-malleability, but it is sufficient for some applications (e.g., [Sah99, DDO+01, Lin03b]). The protocol of this section is not used in any other place of this work. We include it here because it is a simple protocol that demonstrates our techniques. Its main drawback is that we do not know a simple generalization for it to the non-uniform model. Another small drawback is that it uses the subexponential hardness assumption in a much more essential way than our other constructions.

In this subsection (and only in this subsection) we will assume that there exists an evasive set R with the following additional property: there exists some $\epsilon > 0$ such that for every $x \in \{0,1\}^{n^\epsilon}$, the uniform distribution on $x \circ \{0,1\}^{n-n^\epsilon} \cap R_n$ is computationally indistinguishable from $x \circ U_{n-n^\epsilon}$, where R_n denotes $R \cap \{0,1\}^n$ and $\circ$ denotes the string concatenation operator. Note that under suitable assumptions, a variant of the construction of the previous section satisfies this additional property. (Since a random dense enough subset of $\{0,1\}^n$ will satisfy this property.)

Protocol 6.2.3 is our simulation-sound zero-knowledge proof system. We now sketch its analysis. Firstly, we note that, unlike all other proof systems considered in this work, the soundness condition of this system does *not* hold with respect to *non-uniform* polynomial-sized cheating provers. This might have an element $x \circ r \in R$ with $x \notin L$ "hardwired" into it. However this system is sound with respect to *uniform* polynomial-time provers, since such provers cannot sample an element from R.

Let C be a man-in-the-middle adversary for Protocol 6.2.3 that utilizes the synchronizing scheduling. To simulate C we will simply use the simulator for the zero-knowledge proof in the obvious way. That is, the simulator will simulate the first step by following the honest left strategy (i.e., send a random $r \leftarrow_{\mathrm{R}} \{0,1\}^{n-n^\epsilon}$) and then treat the adversary C and the right party as one combined verifier, and simulate the zero-knowledge proof with respect to this verifier. The output of this simulator will be indistinguishable from the view of C in a real interaction. However, because this simulator effectively "rewinds" the right party, it is not at all clear that soundness of the right session is preserved. To show that this is the case we construct an "auxiliary simulator". On input

x, this "auxiliary simulator" will run in super-polynomial time to compute a random r such that $x \circ r \in R$. It will then use the *honest prover* algorithm to prove that either $x \in L$ or $x \circ r \in R$. The simulation soundness property holds for this "auxiliary simulator" because of the evasiveness property of the set R: if the adversary does not copy exactly x and r to the right session, then since it cannot find another $\tilde{x}, \tilde{r}$ such that $\tilde{x} \circ \tilde{r}$ is in the set R, and since no rewinding of the right party is done during the simulation, it is forced to choose a statement $\tilde{x}$ such that $\tilde{x} \in L$. Now the output of the auxiliary simulator is indistinguishable from the output of the real simulator by 2^{n^δ}-time algorithm for some $\delta > 0$. If we scale the security parameter appropriately, we can ensure that deciding membership in L can be done in time less than 2^{n^δ} and therefore the simulation soundness condition must also hold for the real simulator.

Reflection. It is not hard to construct simulation-sound (or even non-malleable) zero-knowledge in a setting where all parties have access to some public trusted verification key of some signature scheme.[17] However, in our setting we obviously do not have access to any such key. In some sense one may view the construction of this section as bypassing this difficulty by using some form of a "keyless signature scheme". That is, one can view a string r such that $x \circ r \in R$ as a *signature* on the string x. Like a (one-time) signature, given a signature $x \circ r \in R$ it is hard to come up with a different signature $x' \circ r' \in R$. However, unlike standard signature schemes, the signing algorithm does not use knowledge of a private key (since no such key exists) but rather uses super-polynomial time.

6.2.3 The Actual Construction

Our non-malleable coin-tossing protocol is Protocol 6.2.4 (See Page 115).

As a first observation, note that R_n is a set that is hard to hit by probabilistic polynomial-time algorithms. Therefore for any such algorithm that plays the left side, with overwhelming probability, it will *not* be the case that $r \in R_n$. Thus when the right side is honest, the soundness of the zero-knowledge argument guarantees that with high probability $r = r_1 \oplus r_2$.[19] The reason that we need to use a universal argument (rather than a standard zero-knowledge proof or argument system for **NP**) is that we are not guaranteed by Theorem 6.2.2 that $R \in \mathbf{NP}$, but rather only that $R \in \mathbf{Dtime}(2^{n^3})$.

It is not hard to verify that the strategies for both the left and right parties can be carried out by probabilistic polynomial-time algorithms. (Note that we use here the prover efficiency condition of universal arguments , inherited from CS proofs.)

In order to show that Protocol 6.2.4 is a secure non-malleable coin-tossing protocol as per Definition 6.1.3, one needs to show that for every adversary C (that uses the *synchronizing* scheduling) there exists an ideal adversary $\widehat{C}$ that simulates the execution of C in a MIM attack. Indeed, let C be a probabilistic polynomial-time algorithm, and consider the execution of Protocol 6.2.4 in the man-in-the-middle setting where C plays the part of the channel. This execution is depicted in Figure 6.4 (Page 6.4). Note that we use the tilde (i.e., $\tilde{}$) symbol to denote the messages of the right session. The messages sent by the right and left parties are computed according to the protocol while the messages sent by the channel are computed by C who may use any (efficiently computable) function of the previous messages.

[17] Indeed, see Protocol 6.4.4 for such a construction.

[18] This is similar to the coin-tossing protocol of [Lin01].

[19] Note that this will *not* be true if we allow the adversary to be a polynomial-sized circuit, that may have an element of R_n hardwired into it.

Public input: 1^n: security parameter	$1^n \downarrow$ $\boxed{L}$ $\boxed{R}$
Steps L,R1.x (Commitment to r_1): Left party selects $r_1 \leftarrow_{\mathrm{R}} \{0,1\}^n$ and commits to it using a perfectly-binding commit-with-extract scheme. We denote the transcript of the commitment by τ_1. We let s_1 denote the randomness used by left party during this step.	$\boxed{\mathsf{Comm\text{-}Ext}(r_1; s_1)} \Rightarrow$
Step R2 (Send r_2): The right party selects a string $r_2 \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends it.	$\xleftarrow{r_2 \leftarrow_{\mathrm{R}} \{0,1\}^n}$
Step L3 (Send r): The left party sends the value $r = r_1 \oplus r_2$. We stress that the left party does not reveal the decommitment of τ_1.[18])	$\xrightarrow{r = r_1 \oplus r_2}$
Steps L,R4.x (Prove that $r = r_1 \oplus r_2$): The left party proves, using a zero-knowledge universal-argument ($ZKUARG$), that either $r = r_1 \oplus r_2$ (where r_1 is the unique string committed to by the transcript τ_1) or $r \in R_n$.	$\boxed{ZKUARG;\ r = r_1 \oplus r_2\ \textbf{or}\ r \in R_n} \Rightarrow$
The result of the protocol is the string r. We will use the convention that if one of the parties aborts (or fails to provide a valid proof) then the other party determines the result of the protocol.	

The right column contains a schematic description of the protocol as defined in the left column.

Protocol 6.2.4. A non-malleable coin-tossing protocol for uniform adversaries

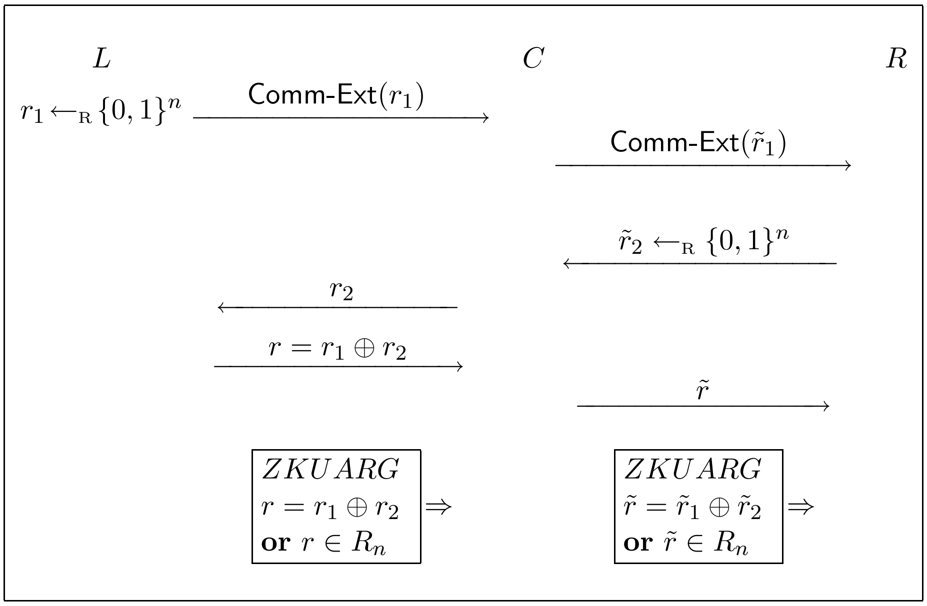


Figure 6.4: Execution of C in the man-in-the-middle setting

We need to simulate C by an "ideal" adversary $\widehat{C}$. The ideal adversary $\widehat{C}$ gets as input $r^{(1)}, r^{(2)}$ and should have two outputs (b, τ). Recall that $b \in \{1, 2\}$ and τ is a string simulating the output of C. Without loss of generality we can assume that τ should simulate the *view* of C in the execution. Clearly this view includes the strings $r, \tilde{r}$ where r is the result of the coin-tossing protocol in the left session and $\tilde{r}$ is the result of the coin-tossing protocol in the right session. For the simulation to be successful, it must hold that $r = r^{(1)}$ and $\tilde{r}$ is either equal to $r^{(1)}$ or to $r^{(2)}$. If this holds then we can decide by examining τ whether b should equal 1 or 2 based on whether $\tilde{r} = r^{(1)}$ or $\tilde{r} = r^{(2)}$. We see that the output b is redundant and that to prove that Protocol 6.2.4 is a secure non-malleable coin-tossing protocol it is enough to prove the following theorem:

Theorem 6.2.5. *Suppose that C is a probabilistic polynomial-time algorithm describing the strategy for a synchronizing adversary for Protocol 6.2.4, then there exists an algorithm $\widehat{C}'$ (computable by a probabilistic expected polynomial-time Turing machine with oracle access to C) with a single output such that, if $r^{(1)}, r^{(2)}$ are chosen uniformly and independently in $\{0,1\}^n$ then,*

1. *$\widehat{C}'(r^{(1)}, r^{(2)})$ is computationally indistinguishable from the view of C in a real execution of Protocol 6.2.4 in the man-in-the-middle setting.*
2. *Let $r, \tilde{r}$ be the result of the coin-tossing algorithm in the left and right sessions recorded in the transcript $\widehat{C}'(r^{(1)}, r^{(2)})$. Then, with overwhelming probability it is the case that $r = r^{(1)}$ and $\tilde{r} \in \{r^{(1)}, r^{(2)}\}$.*

6.2.4 Proof of Theorem 6.2.5

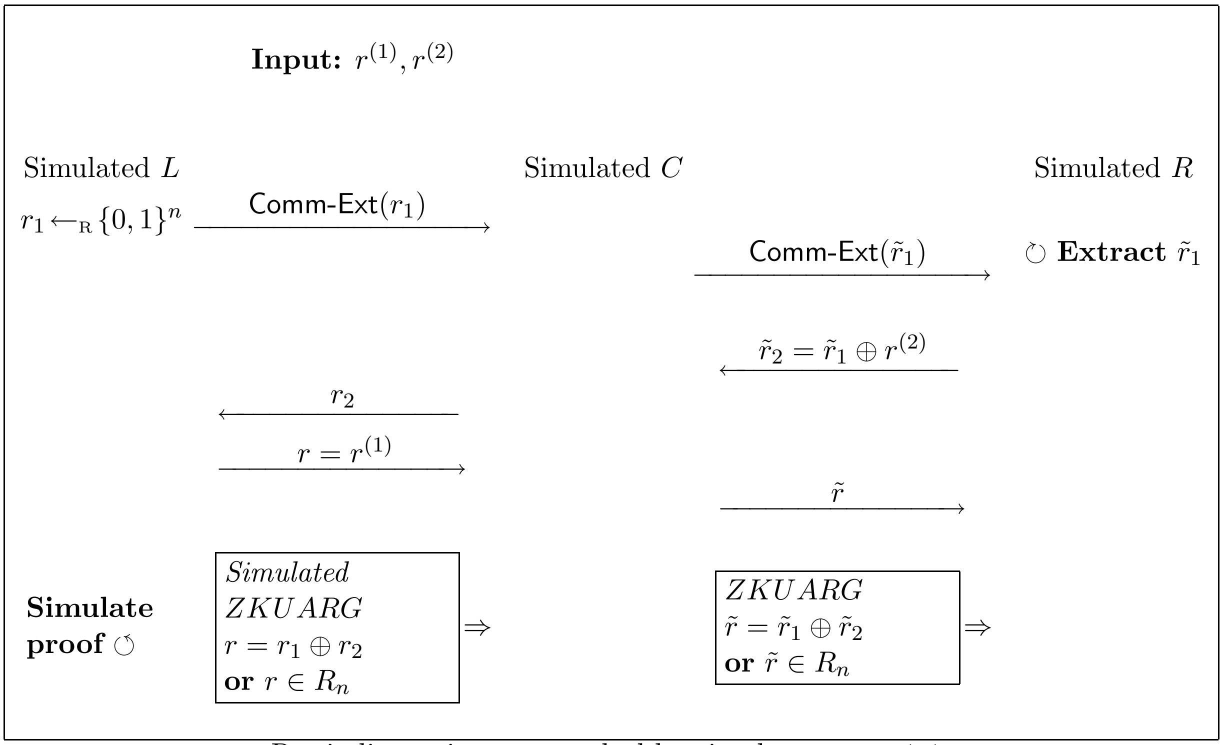
Let C be a probabilistic polynomial-time algorithms representing the strategy of a synchronizing adversary for Protocol 6.2.4. In order to prove Theorem 6.2.5 we need to construct an algorithm $\widehat{C}'$ that simulates C. Figure 6.5 (Page 117) contains a schematic description of Algorithm $\widehat{C}'$.

Operation of Algorithm $\widehat{C}'$. Algorithm $\widehat{C}'$ gets as input two strings $r^{(1)}, r^{(2)}$ that were chosen uniformly and independently at random and in addition it gets black-box access to algorithm C. Algorithm $\widehat{C}'$ emulates C's execution by running the honest left and right strategy, with some modifications as described below.

Algorithm $\widehat{C}'$ departs from the honest strategy in the right session (Steps L,R1.x) where it uses the *commitment extractor* of the commit-with-extract scheme Comm-Ext to simulate the execution of this phase, while also obtaining the value $\tilde{r}_1$ committed to by C. Note that in order to do that, $\widehat{C}'$ needs to treat both C and the left party L as a single combined sender for the commitment scheme and rewind them both at the same time.

The next modification is that instead of choosing $\tilde{r}_2$ to be a random string as is done by the honest right party, Algorithm $\widehat{C}'$ uses $\tilde{r}_2 = \tilde{r}_1 \oplus r^{(2)}$. Note however that this is still a uniform string because $r^{(2)}$ is uniformly distributed. In the left session in Step L3 $\widehat{C}'$ sends $r = r^{(1)}$ instead of $r = r_1 \oplus r_2$. This means that with high probability the statement $r = r_1 \oplus r_2$ is *false*. Also, with high probability it also holds that $r \notin R_n$. However, $\widehat{C}'$ uses the *simulator* for the $ZKUARG$ in order to simulate a proof for the false statement $r = r_1 \oplus r_2$ or $r \in R_n$. Similarly to the case when using the commitment extractor, Algorithm $\widehat{C}'$ will need to treat the adversary C and the honest right party as one combined verifier algorithm. Thus Algorithm $\widehat{C}'$ will also rewind the honest right party in this step. Note that all this can be carried out in expected polynomial-time.[20] Note that

[20]Actually we can also have a *strict* polynomial-time non-black-box simulator by using the protocols of [Bar01] and [BL02].



Rewinding points are marked by circular arrows ↺, ↻

Figure 6.5: Algorithm $\widehat{C}'$ – simulation of C

Algorithm $\widehat{C}'$ would not be well-defined if we needed to invoke the simulator and extractor at the same time, since it would mean that algorithm $\widehat{C}'$ would be rewinding itself. However, this can not happen since we assume the *synchronizing* scheduling.

Now that algorithm $\widehat{C}'$ is specified all that is left is to prove the two parts of Theorem 6.2.5. It turns out that Part 1 can be proven using the standard hybrid argument (relying on the security of the commitment scheme and zero-knowledge simulator). In contrast, the proof of Part 2 is much more complicated and it is for proving this part that we needed to introduce the evasive set R in the first place. We start with the proof of the more complicated part.

6.2.5 Proof of Theorem 6.2.5 Part 2

From a first impression, by looking at Figure 6.5 one may think that Part 2 of Theorem 6.2.5 should be easy to prove. After all, in the left session it is certainly the case that $r = r^{(1)}$ and the soundness of the universal-argument system used in the right session should ensure us that $\tilde{r} = \tilde{r}_1 \oplus \tilde{r}_2 = r^{(2)}$ (because it is unlikely that C can select $\tilde{r} \in R_n$). However, there is a caveat in this reasoning. The problem is that soundness is only ensured in an interactive setting where the prover only has access to a single interaction with the honest verifier. However, since Algorithm $\widehat{C}'$ is using the simulator in the left session, it will actually rewind also the verifier algorithm of the right party. he fact that Algorithm $\widehat{C}'$ gets the ability to rewind the verifier algorithm ruins our ability to argue about the soundness of the universal-argument system. Indeed, this problem is real (i.e., the soundness of the system may indeed be compromised). For example, consider an adversary that uses the *relaying* content strategy (i.e., copies all messages from the left session to the right session and vice versa). In an execution with such an adversary, it will be the case that $\tilde{r} = r$ and the proof in the right session will pass verification (since for such an adversary, the right session is identical to the left session). Because of the indistinguishability of our simulator, when we simulate the relaying adversary with Algorithm $\widehat{C}'$, also in the simulated transcript it holds that $\tilde{r} = r = r^{(1)}$. However, since $r^{(1)}$ is chosen at random, with overwhelming probability, it will *not* be the case that $\tilde{r} = \tilde{r}_1 \oplus \tilde{r}_2$. Therefore, in the simulated transcript, the statement proven by the adversary in the right session will be *false*. Of course, this does not mean that Part 2 of Theorem 6.2.5 is false. Indeed, in the case of the relaying adversary it holds $\tilde{r} = r^{(1)}$ which is also allowed by the statement of Part 2. It just means that the naive approach to proving this part fails. We now turn to the actual proof.

We assume, for the sake of contradiction, that there exists a probabilistic polynomial-time algorithm C such that with non-negligible probability the corresponding ideal adversary $\widehat{C}'$ outputs a transcript where the result of right session $\tilde{r}$ is neither $r^{(1)}$ nor $r^{(2)}$. Note that in this case it must hold that the proof in the right session passes verification (or otherwise by our convention the right party can simply choose $\tilde{r} = r^{(2)}$). We consider the following $2^{O(n^3)}$-time algorithm $\widehat{C}''$ (depicted in Figure 6.6). Algorithm $\widehat{C}''$ behaves almost exactly as $\widehat{C}'$ with two differences:

1. In the left session (in Step L3) it chooses $r \leftarrow_{\mathrm{R}} R_n$ instead of choosing $r = r^{(1)}$ (where $r^{(1)} \leftarrow_{\mathrm{R}} \{0,1\}^n$). (This takes $2^{O(n^3)}$ steps using the constructibility property of R.)
2. Then, in the $ZKUA$ phase (Steps L,R4.x) Algorithm $\widehat{C}''$ does not use the zero-knowledge simulator but rather the *honest prover* algorithm of the universal argument to prove the true statement that either $r = r_1 \oplus r_2$ or $r \in R_n$. (This takes $2^{O(n^3)}$ steps.)

The output of $\widehat{C}''$ is computationally indistinguishable (by uniform algorithms) from the output of $\widehat{C}'$, even if the distinguisher is given $r^{(2)}$ (but not $r^{(1)}$). Indeed, this follows from the pseudorandomness of R_n and the zero-knowledge property of the universal-argument. Yet, combined with

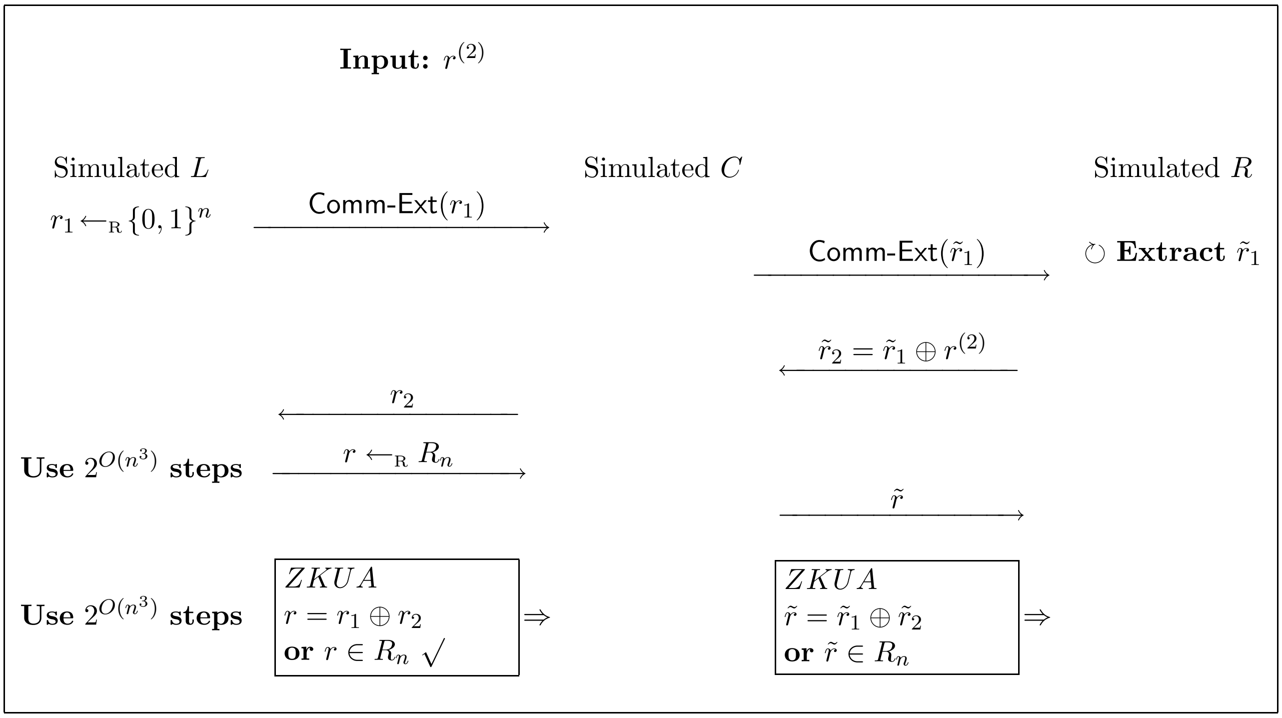


Figure 6.6: Algorithm $\widehat{C}''$

the hypothesis that in the output of $\widehat{C}'$ with non-negligible probability $\tilde{r} \notin \{r^{(1)} = r, r^{(2)}\}$, this implies that in the output of $\widehat{C}''$, with non-negligible probability it is the case that the following three conditions hold simultaneously:

1. $\tilde{r} \neq r$
2. $\tilde{r} \neq r^{(2)} = \tilde{r}_1 \oplus \tilde{r}_2$
3. The proof that either $\tilde{r} = \tilde{r}_1 \oplus \tilde{r}_2$ or $\tilde{r} \in R_n$ passes verification.

Now the soundness property of the universal arguments holds against 2^{n^5}-sized circuits. Since we are using less time than this and no rewinding/simulation is done during the relevant time[21] by $\widehat{C}''$ it can only be with negligible probability that the proof passes verification and the statement is false. This means that with non-negligible probability it will be the case that $\tilde{r} \neq r$ and $\tilde{r} \in R_n$ (since $\tilde{r} \neq \tilde{r}_1 \oplus \tilde{r}_2$). Now suppose that we halt the execution of $\widehat{C}''$ at the point where $\tilde{r}$ is computed. Up to this point $\widehat{C}''$ only needs to use a polynomial number of steps if it is given as input a random element $r \leftarrow_{\mathrm{R}} R_n$. This means that we have a probabilistic polynomial-time algorithm that gets a string $r \leftarrow_{\mathrm{R}} R_n$ and outputs a string $\tilde{r}$ that with non-negligible probability will be both different from r and a member of R_n. But this is clearly a contradiction to the evasiveness property of the set R_n. $\square$

6.2.6 Proof of Theorem 6.2.5 Part 1

We will now prove Part 1 of Theorem 6.2.5. That is, we prove the following claim:

[21] $\widehat{C}''$ does use rewinding in the extraction stage but this is done before the $ZKUA$ phase. Also in the extraction phase $\widehat{C}''$ only needs to rewind the honest left algorithm and not the honest right algorithm.

Claim 6.2.6. *Let C be a probabilistic polynomial-time adversary strategy for Protocol 6.2.4. Let $\widehat{C}'$ be the simulator for C, as described in Section 6.2.4. Then, $\widehat{C}'(r^{(1)}, r^{(2)})$, where $r^{(1)}, r^{(2)} \leftarrow_R \{0,1\}^n$, is computationally indistinguishable from the view of C in a real execution of Protocol 6.2.4 in the man-in-the-middle setting.*

We prove the claim using the hybrid argument. We let $\mathbf{H}_0$ be the view of C in a real execution, and we let $\mathbf{H}_4$ be $\widehat{C}'(r^{(1)}, r^{(2)})$, where $r^{(1)}, r^{(2)} \leftarrow_{\mathrm{R}} \{0,1\}^n$. We prove that $\mathbf{H}_0 \equiv_{\mathrm{C}} \mathbf{H}_4$ by showing three intermediate random variables $\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3$ such that

$$\mathbf{H}_0 \equiv_{\mathrm{C}} \mathbf{H}_1 \equiv_{\mathrm{C}} \mathbf{H}_2 \equiv_{\mathrm{C}} \mathbf{H}_3 \equiv_{\mathrm{C}} \mathbf{H}_4$$

We now describe these intermediate random variables, and show that indeed for every $1 \leq i \leq 4$, $\mathbf{H}_i$ is computationally indistinguishable from $\mathbf{H}_{i-1}$. To describe the random variable $\mathbf{H}_i$, we will describe the differences between it and the variable $\mathbf{H}_{i-1}$, and then show that the two variables are indistinguishable.

Hybrid $\mathbf{H}_1$: Simulated ZKUA. The difference between $\mathbf{H}_1$ and $\mathbf{H}_0$ is that in $\mathbf{H}_1$ we use the *simulator* of the zero-knowledge universal argument of Steps L,R4.x to simulate the view of the adversary C in these steps in the left session. Note that the verifier that is simulated is the combined strategy of C and the honest Right algorithm in these steps. The two random variables are indistinguishable by the zero-knowledge condition of the universal argument.

Hybrid $\mathbf{H}_2$: Choose $r = r^{(1)}$. The difference between $\mathbf{H}_2$ and $\mathbf{H}_1$ is that in $\mathbf{H}_2$ we choose the string r sent by the Left party in Step R2 of the left session to be $r^{(1)}$, where $r^{(1)}$ is chosen at random from $\{0,1\}^n$, instead of choosing r to be equal to $r_1 \oplus r_2$. The distributions $\mathbf{H}_2$ and $\mathbf{H}_1$ are computationally indistinguishable by the security of the commitment scheme used in Steps L,R1.x. (Note that in both hybrids, the coins used in the commitment scheme of these steps are not used anywhere else in the execution.)

Hybrid $\mathbf{H}_3$: Extract commitment. The difference between $\mathbf{H}_3$ and $\mathbf{H}_2$ is that in $\mathbf{H}_3$ we use the *commitment extractor* of the commit-with-extract scheme of Steps L,R1.x to simulate the view of the adversary C in these steps in the right session. Note that the sender that is simulated is the combined strategy of C and the honest Left algorithm in these steps. The two random variable are indistinguishable by the simulation condition of the commit-with-extract scheme (see Definition 2.5.2). Note that in addition to outputting a simulated view, the commitment extractor also outputs as an auxiliary outputs the value $\tilde{r}_1$ that is committed to by the adversary in this simulated view.

Hybrid $\mathbf{H}_4$: Send $\tilde{r}_2 = \tilde{r}_1 \oplus r^{(2)}$. The difference between $\mathbf{H}_4$ and $\mathbf{H}_3$ is that in $\mathbf{H}_4$, in Step R2 of the right session, we choose $\tilde{r}_2$ to be $\tilde{r}_1 \oplus r^{(2)}$, where $\tilde{r}_1$ is the value extracted by the commitment extractor, and $r^{(2)}$ is chosen at random. We note that even though we now choose $\tilde{r}_2$ in a different way, its distribution is still the uniform distribution (since $r^{(2)}$ is chosen at random). Therefore, the two variables $\mathbf{H}_4$ and $\mathbf{H}_3$ are identically distributed.

The proof is finished by observing that $\mathbf{H}_4$ is indeed the random variable $\widehat{C}'(r^{(1)}, r^{(2)})$, where $r^{(1)}, r^{(2)} \leftarrow_{\mathrm{R}} \{0,1\}^n$.

□

6.3 Dealing with Non-Uniform Adversaries

In this section, we show how to modify Protocol 6.2.4 to obtain security even against adversaries that use *non-uniform* algorithms. Specifically, under the same assumptions we will construct a non-malleable coin-tossing protocol secure against *polynomial-sized* circuits (rather than probabilistic polynomial-time *uniform* algorithms ,as was done in Section 6.2). The protocol and its simulator will be quite similar to the uniform case. However, our proof of security will be somewhat different and will involve a *non-black-box* use of the adversary's code.

6.3.1 Evasive Set Families

An important component in our construction is a generalization of evasive sets as defined in Section 6.2.1, called *evasive set family.* Roughly speaking, an evasive set family is a family of sets indexed by strings, where each set is evasive (in the sense of Definition 6.2.1) with respect to algorithms that get its index as an advice string. The formal definition follows:[22]

Definition 6.3.1 (Evasive set family). Let $\{R_\alpha\}_{\alpha\in\{0,1\}^*}$ be family of sets, where for any $\alpha \in \{0,1\}^*$, $R_\alpha \subseteq \{0,1\}^*$. Denote $R_{\alpha,n} \overset{def}{=} R_\alpha \cap \{0,1\}^n$. We say that the family $\{R_\alpha\}_{\alpha\in\{0,1\}^*}$ is an *evasive set family* if the following conditions hold with respect to some negligible function $\mu(\cdot)$:

Constructibility: There exists a Turing machine M such that $M(1^n,\alpha)$ runs for $|\alpha|2^{n^3}$ steps and outputs a list of all the elements in the set $R_{\alpha,n}$. In particular this means that deciding whether or not $r \in R_{\alpha,n}$ can be done in time $|\alpha|2^{|r|^3}$.

Pseudorandomness: The set $R_{\alpha,n}$ is pseudorandom against uniform probabilistic $n^{O(\log n)}$-time algorithms with advice α. That is, for any probabilistic polynomial-time Turing machine M it holds that

$$\left| \Pr_{r\leftarrow_{\mathrm{R}} R_{\alpha,n}}[M(\alpha,r)=1] - \Pr_{r\leftarrow_{\mathrm{R}}\{0,1\}^n}[M(\alpha,r)=1] \right| < \mu(n)$$

Evasiveness: For any probabilistic $n^{O(\log n)}$-time Turing machine M and for any $r \in R_{\alpha,n}$,

$$\Pr[M(\alpha,r) \in R_{\alpha,n} \setminus \{r\}] < \mu(n)$$

Equivalent strings. Intuitively, we say that two strings α,α' are *equivalent* if each string can be computed from the other string by a uniform "efficient"[23] algorithms. More precisely, we define the notion of equivalence as follows:

Definition 6.3.2 (Equivalent strings). Let $\mu : \mathbb{N} \to [0,1]$ be some function. Two strings $\alpha,\alpha' \in \{0,1\}^*$ are *μ-equivalent* if there exist two probabilistic algorithms M and M' with description size at most $\log\mu(n)$ and running-time at most $\frac{1}{\mu(n)}$ (where $n = |\alpha| + |\alpha'|$) such that both

$$\Pr[M(\alpha) = \alpha'] > \mu(n)$$

[22] We have chosen to define evasive set families with respect to $n^{O(\log n)}$-time algorithms instead of polynomial-time. This change somewhat simplifies our exposition but is not really significant. In particular, we could have used any other fixed super-polynomial function.

[23] For convenience, we define "efficient" in this section as running in time $n^{O(\log n)}$. See Footnote 22.

and

$$Pr[M'(\alpha') = \alpha] > \mu(n)$$

hold.

We say that α and α' are *equivalent* if they are $n^{-\log^2 n}$-equivalent. We say that an evasive set family is *nice* if $R_\alpha = R_{\alpha'}$ whenever α and α' are equivalent.

As in the case of evasive sets, under the assumptions of this paper there exists an evasive set family. That is, we have the following theorem:

Theorem 6.3.3. *Suppose that 2^{n^ϵ}-strong one-way functions exist, then there exists an evasive set family. Furthermore, this family is nice.*

Proof Sketch: The proof of Theorem 6.3.3 is almost identical to the proof of Theorem 6.2.2, since once a string α is fixed, one can enumerate and execute in $n^{\mathrm{polylog}(n)}$ time all Turing machines of $\log^3 n$-sized description and $n^{\log^3 n}$ probabilistic running time that get α as an advice string. Using this, one can construct for each α, a set R_α that is evasive and pseudorandom with respect to such machines.

We can transform a family $\{R_\alpha\}$ to a family $\{R_{\alpha'}\}$ that is nice in the following way. To compute $R_{\alpha'}$ we first find in $n^{\mathrm{polylog}(n)}$-time the lexicographically first string α such that α is equivalent to α' and then compute R_α.[24] □

6.3.2 The construction

Protocol 6.3.4 is our non-malleable coin-tossing protocol for non-uniform adversaries. It is very similar to the non-malleable coin-tossing protocol for uniform adversaries (Protocol 6.2.4). In fact, there are only two main changes:

First modification (adding a preliminary stage). We add a preliminary stage (Steps L,R 0.x) where each party sends a commitment to a hash of the all-zeros string and prove that it knows the hashed value using a zero-knowledge universal argument ($ZKUA$).[25]

Step L0.1.x - left commitment The left player commits to a hash of the all-zeros string and proves knowledge of the hashed value.

Step R0.1.1 (Right sends hash): Right party chooses a random collision-resistent hash function h_1 and sends it to the left party.

Step L0.1.2 (Left commits): The left party a commitment to $h_1(0^n)$. That is, it chooses a string s_1 (random coins for the commitment) and sends $y_1 = \mathsf{Com}(h_1(0^n); s_1)$

[24] There is a slight subtlety here. When testing equivalence between α and α' we will need to use a pseudorandom generator to check the condition that $\Pr[M(\alpha) = \alpha'] > n^{-\log^2 n}$ for all machines M of appropriate description and running time. This means that we cannot compute *exactly* whether two strings are equivalent. To fix this, we will consider strings as equivalent as long as the probability estimated using the pseudorandom generator is at least $n^{-\log^3 n}$. This will ensure that if α' and α'' are equivalent as per Definition 6.3.2, then the lexicographically first string α we find will be identical for both of them.

[25] We do not use a commit-with-extract scheme here because we need to prove knowledge of the hash's preimage and not knowledge of the value that is committed to.

Steps L,R0.1.2.x (Left proves knowledge): The left party proves using a $ZKUA$ that it knows a string α (where $|\alpha| \leq n^{\log n}$) and a string s_1 such that $y_1 = \mathsf{Com}(h_1(\alpha); s_1)$ where y_1 is the commitment sent at Step L0.1.1

Note: We stress that since we use a universal argument the length of the string α is *not* bounded by any fixed polynomial in the security parameter. However for convenience, we will require that $|\alpha| \leq n^{\log n}$. Note that if the left party follows the protocol then it would be the case that $\alpha = 0^n$.

Steps L,R0.2.x - right commitment These steps are almost a "mirror image" of Steps L,R0.1.x (with one slight difference). That is, the Right player now commits to a hash of the all-zeros string and proves knowledge of the hashed value.

Step R0.2.1 (Right commits): The right party chooses a random collision resistant hash function h_2 and sends h_2 along with a commitment to $h_2(0^n)$. That is, it chooses a string s_2 (random coins for the commitment) and sends h_2 and $y_2 = \mathsf{Com}(h_2(0^n); s_2)$.[26]

Steps L,R0.2.2.x (Right proves knowledge): The right party proves using a ZKUA that it knows a string β (where $|\beta| \leq n^{\log n}$) and a string s_2 such that $y_2 = \mathsf{Com}(h_2(\alpha); s_2)$ where y_2 is the commitment sent at Step R0.2.1

Second modification (modifying the $ZKUA$). In Step 5, the left party proves using the zero-knowledge universal-argument that either $r = r_1 \oplus r_2$ or $r \in R_{\alpha\circ\beta,n}$ where α is such that $y_1 = \mathsf{Com}(h_1(\alpha), s_1)$ for some s_1 and β is such that $y_2 = \mathsf{Com}(h_2(\beta), s_2)$ for some s_2. (Recall that $\alpha \circ \beta$ denotes the concatenation of α and β.)

By applying these changes we obtain Protocol 6.3.4 (see Page 124).

As in the uniform case (i.e., of Protocol 6.2.4), what we need to prove is the following Theorem (which is the non-uniform analog of Theorem 6.2.5):

Theorem 6.3.5. *Let $C_{\mathsf{n.u.}}$ be a polynomial-sized circuit (representing the adversary's strategy for Protocol 6.3.4). Then, there exists an algorithm $\widehat{C}'_{\mathsf{n.u.}}$ (that can be computed in expected probabilistic polynomial-time algorithm with oracle access to $C_{\mathsf{n.u.}}$) such that if $r^{(1)}, r^{(2)}$ are chosen uniformly and independently in $\{0,1\}^n$ then:*

1. *$\widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)})$ is computationally indistinguishable from the view of $C_{\mathsf{n.u.}}$ in a real execution of Protocol 6.3.4 with the honest left and right parties.*

2. *Let $r, \tilde{r}$ be the result of the coin-tossing protocol in the left and right sessions of the view $\widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)})$. Then, with overwhelming probability it is the case that $r = r^{(1)}$ and $\tilde{r} \in \{r^{(1)}, r^{(2)}\}$.*

[26] Note that this step is not an exact mirror image of the previous step since the Right party, that is the *sender* of the commitment, is choosing the hash function.

Public input: 1^n: security parameter	1^n $\downarrow$ $\boxed{L}$ $\boxed{R}$
Steps L,R0.1.x (Left commits to α): Right party chooses and sends a hash h_1. Left sends $y_1 = \mathsf{Com}(h_1(0^n))$ and then proves using a ZKUA that it knows a value α such that $y_1 = \mathsf{Com}(h_1(\alpha))$ (where $\lvert\alpha\rvert \le n^{\log n}$). **Steps L,R0.1.x (Right commits to β):** Right party chooses a hash h_2, and sends h_2 and $y_2 = \mathsf{Com}(h_2(0^n))$. It then proves using a ZKUA that it knows a value β such that $y_2 = \mathsf{Com}(h_2(\beta))$ (where $\lvert\beta\rvert \le n^{\log n}$).	$\xleftarrow{h_1}$ $\xrightarrow{h_1, y_1 = \mathsf{Com}(h_1(0^n))}$ $\boxed{\begin{array}{l} ZKUA \text{ of } \alpha, s \text{ s.t.} \\ y_1 = \mathsf{Com}(h_1(\alpha), s) \end{array}} \Rightarrow$ $\xleftarrow{h_2, y_2 = \mathsf{Com}(h_2(0^n))}$ $\Leftarrow \boxed{\begin{array}{l} ZKUA \text{ of } \beta, s \text{ s.t.} \\ y_2 = \mathsf{Com}(h_2(\beta), s) \end{array}}$
Continue as in Protocol 6.2.4 (The only change is in Steps L,R4.x)	
Steps L,R1.x (Commitment to r_1): *(unchanged)* Left party selects $r_1 \leftarrow_{\mathrm{R}} \{0,1\}^n$ and commits to it using a perfectly-binding commit-with-extract scheme. We denote the transcript of the commitment by τ_1. We let s_1 denote the randomness used by left party during this step. **Step R2 (Send r_2):** *(unchanged)* The right party selects a string $r_2 \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends it. **Step L3 (Send r):** *(unchanged)* The left party sends the value $r = r_1 \oplus r_2$ (without revealing the decommitment of τ_1). **Steps L,R4.x (Prove that $r = r_1 \oplus r_2$):** The left party proves using a zero-knowledge universal-argument ($ZKUA$) that either $r = r_1 \oplus r_2$ or $r \in R_{\alpha\circ\beta,n}$, where $y_1 = \mathsf{Com}(h_1(\alpha))$ and $y_2 = \mathsf{Com}(h_2(\beta))$.	$\boxed{\mathsf{Comm\text{-}Ext}(r_1; s_1)} \Rightarrow$ $\xleftarrow{r_2 \leftarrow_{\mathrm{R}} \{0,1\}^n}$ $\xrightarrow{r = r_1 \oplus r_2}$ $\boxed{\begin{array}{l} ZKUA \\ r = r_1 \oplus r_2 \\ \textbf{or } r \in R_{\alpha\circ\beta,n} \end{array}} \Rightarrow$
The result of the protocol is the string r. We will use the convention that if one of the parties aborts (or fails to provide a valid proof) then the other party determines the result of the protocol.	

Protocol 6.3.4. A non-malleable coin-tossing protocol for non-uniform adversaries

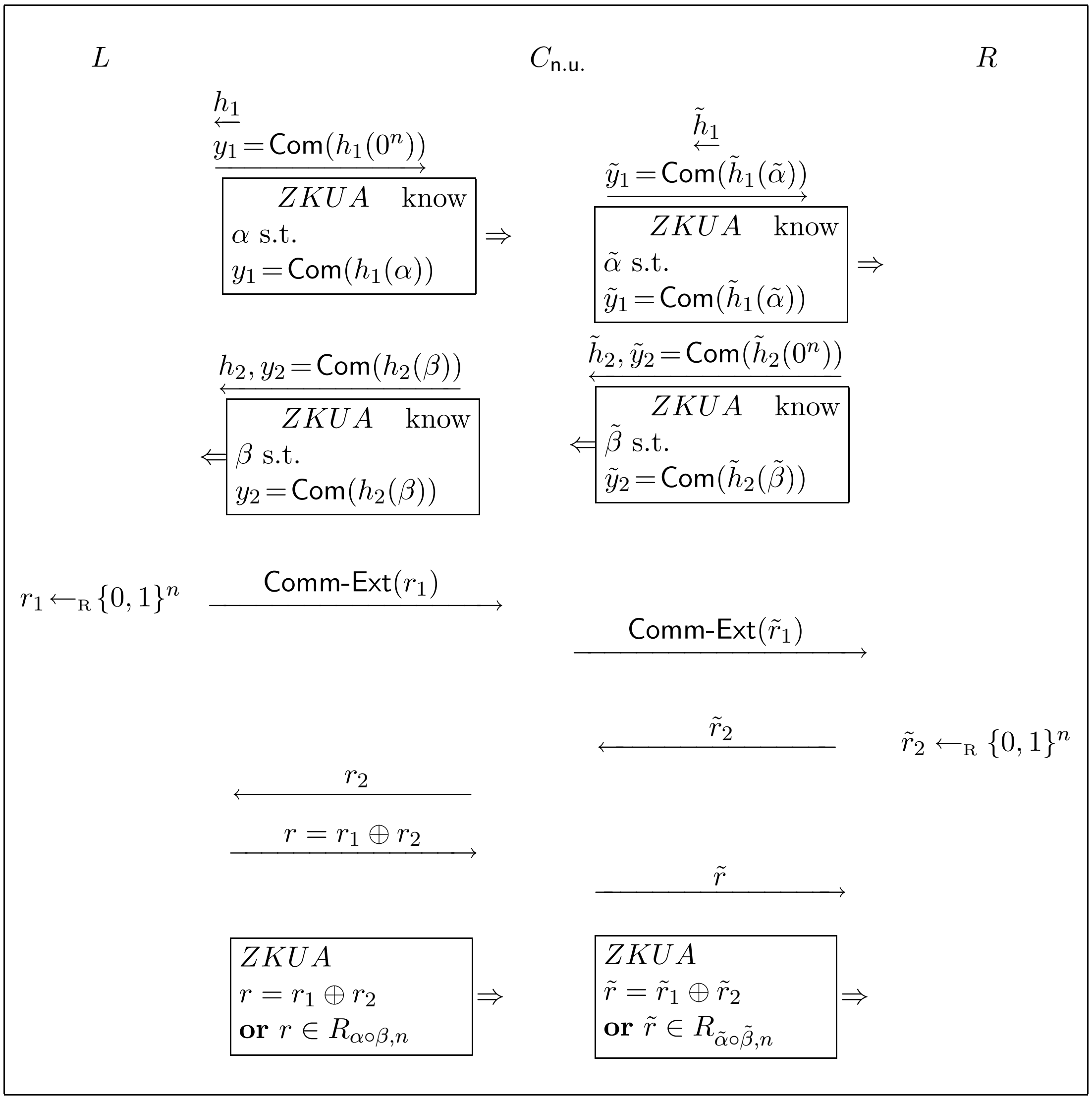


Figure 6.7: Execution of $C_{\mathsf{n.u.}}$ in the man-in-the-middle setting

6.3.3 Proof of Theorem 6.3.5

To prove Theorem 6.3.5, consider a polynomial-sized circuit $C_{\mathsf{n.u.}}$ and consider the execution of Protocol 6.3.4 with $C_{\mathsf{n.u.}}$ playing the part of the channel. Such an execution is depicted in Figure 6.7.

We will use a simulator $\widehat{C}'_{\mathsf{n.u.}}$ that is very similar to the simulator used in the uniform case (see Section 6.2.4). In fact, the only change will be that we need to simulate also the preliminary steps (Steps L,R0.x). To simulate these steps, algorithm $\widehat{C}'_{\mathsf{n.u.}}$ will simply follow the honest left and right strategy (i.e., commit to a hash of the all zeros string). To simulate the other steps, we will follow the same strategy as the simulator $\widehat{C}'$, as described in Section 6.2.4 (see also Figure 6.5). That is, we will use the commitment extractor to extract $\tilde{r}_1$, choose $\tilde{r}_2$ to be $\tilde{r}_1 \oplus r^{(2)}$, and use the simulator of the zero-knowledge universal-argument to simulate Steps L,R4.x. Figure 6.8 (Page 126) contains a schematic description of the resulting algorithm $\widehat{C}'_{\mathsf{n.u.}}$.

As in the uniform case, to prove Theorem 6.3.5, we need to prove two Lemmas analogous to Parts 1 and 2 of Theorem 6.2.5:

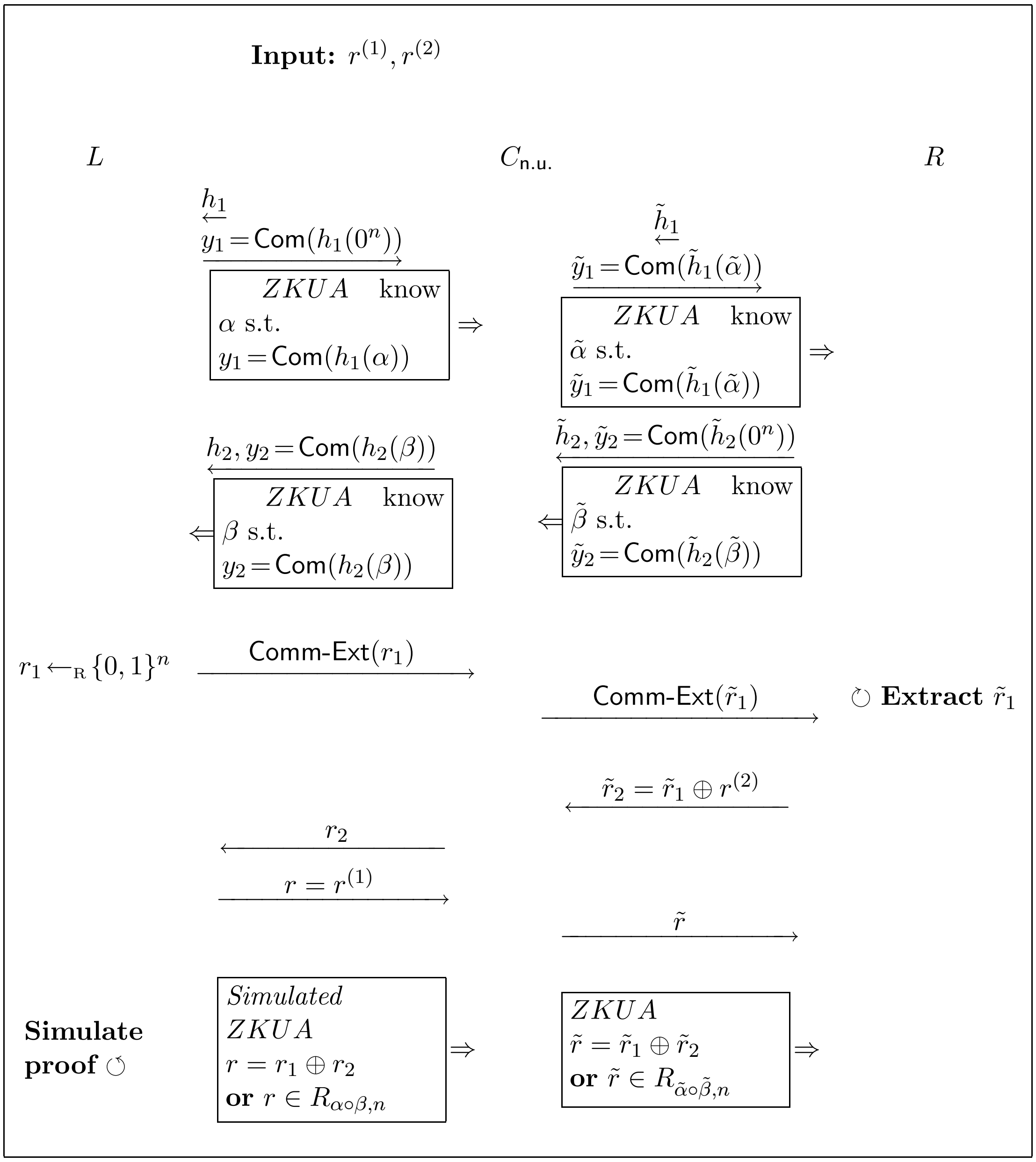


Figure 6.8: Algorithm $\widehat{C}'_{\mathsf{n.u.}}$ - simulation of $C_{\mathsf{n.u.}}$

Lemma 6.3.6. *For $r^{(1)}, r^{(2)} \leftarrow_R \{0,1\}^n$, $\widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)})$ is computationally indistinguishable from the view of $C_{\mathsf{n.u.}}$ when executing Protocol 6.3.4 with the honest L and R in the man-in-the-middle-setting.*

Lemma 6.3.7. *For $r^{(1)}, r^{(2)} \leftarrow_R \{0,1\}^n$, with overwhelming probability $\widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)})$ is a view of an execution of Protocol 6.3.4 where the result r of the left session is $r^{(1)}$ and the result $\tilde{r}$ of the right session is either $r^{(1)}$ or $r^{(2)}$.*

The proof of Lemma 6.3.6 is obtained by fairly standard hybrid arguments, and is almost identical to the proof of Theorem 6.2.5 Part 1.[27] Thus we omit this proof here. In contrast, the proof of Lemma 6.3.7 is more complicated and involves a *non-black-box* use of the code of the adversary $C_{\mathsf{n.u.}}$.

6.3.4 Proof of Lemma 6.3.7

The general outline proof of Lemma 6.3.7 follows the proof of Theorem 6.2.5 Part 2 (see Section 6.2.5). We assume, for the sake of contradiction, that there exists a polynomial-sized circuit $C_{\mathsf{n.u.}}$, such that with non-negligible probability in the transcript $\widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)})$ it is the case that $\tilde{r} \notin \{r^{(1)}, r^{(2)}\}$ (where $\tilde{r}$ denotes the result string in the right session).[28] We now construct a $2^{O(n^3)}$-time algorithm $\widehat{C}''_{\mathsf{n.u.}}$ with one input that will have the following two properties:

1. For randomly chosen $r^{(2)} \leftarrow_{\mathrm{R}} \{0,1\}^n$, the random variable $(r^{(2)}, \widehat{C}''_{\mathsf{n.u.}}(r^{(2)}))$ is computationally indistinguishable by uniform algorithms from the random variable $(r^{(2)}, \widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)}))$, where $r^{(1)}, r^{(2)} \leftarrow_{\mathrm{R}} \{0,1\}^n$.
2. With overwhelming probability it is the case that in the output of $\widehat{C}''_{\mathsf{n.u.}}(r^{(2)})$ either $\tilde{r} = r$ or $\tilde{r} = r^{(2)}$, where r and $\tilde{r}$ denote the result of the coin-tossing protocol in the left and right session, respectively.

As in Section 6.2.5, the existence of an algorithm $\widehat{C}''_{\mathsf{n.u.}}$ satisfying these two properties directly leads to a contradiction. This is because they imply that also in the transcript $\widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)})$ it is the case that with overwhelming probability $\tilde{r} \in \{r, r^{(2)}\}$ (note that this condition can be tested by a uniform algorithm). Since we know that in the transcript $\widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)})$ it holds that $r = r^{(1)}$ we get that $\tilde{r} \in \{r^{(1)}, r^{(2)}\}$ and so the contradiction follows.

What remains to be done is to describe the algorithm $\widehat{C}''_{\mathsf{n.u.}}$ and prove that it satisfies the two properties. Algorithm $\widehat{C}''_{\mathsf{n.u.}}$ is described in Figure 6.9 (Page 128); It follows the operation of Algorithm $\widehat{C}'_{\mathsf{n.u.}}$ with the following changes:

1. In Steps L,R0.1.x of the *left* session, Algorithm $\widehat{C}''_{\mathsf{n.u.}}$ sets the hashed string α to be the *description of $C_{\mathsf{n.u.}}$'s code* instead of $\alpha = 0^n$, which is what is done by the honest left party L and by Algorithm $\widehat{C}'_{\mathsf{n.u.}}$.
2. In Steps L,R0.2.x of the *right* session, Algorithm $\widehat{C}''_{\mathsf{n.u.}}$ follows the "mirror image" of the previous steps. That is, Algorithm $\widehat{C}''_{\mathsf{n.u.}}$ will set the hashed string $\tilde{\beta}$ to be the *description of $C_{\mathsf{n.u.}}$'s code* instead of $\tilde{\beta} = 0^n$ (as is done by the honest right party R and by Algorithm $\widehat{C}'_{\mathsf{n.u.}}$).

[27] This is due to the fact that the only difference between the simulator $\widehat{C}'_{\mathsf{n.u.}}$ we present here and the simulator $\widehat{C}'$ of Section 6.2.4 is that Algorithm $\widehat{C}'_{\mathsf{n.u.}}$ needs to simulates also the preliminary phase of Steps L,R0.x. However, in these steps it uses the same strategy as used by the honest parties.

[28] Note that by the definition of Algorithm $\widehat{C}'_{\mathsf{n.u.}}$ it is always the case that $r = r^{(1)}$, where r denote the result string of the left session in $\widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)})$.

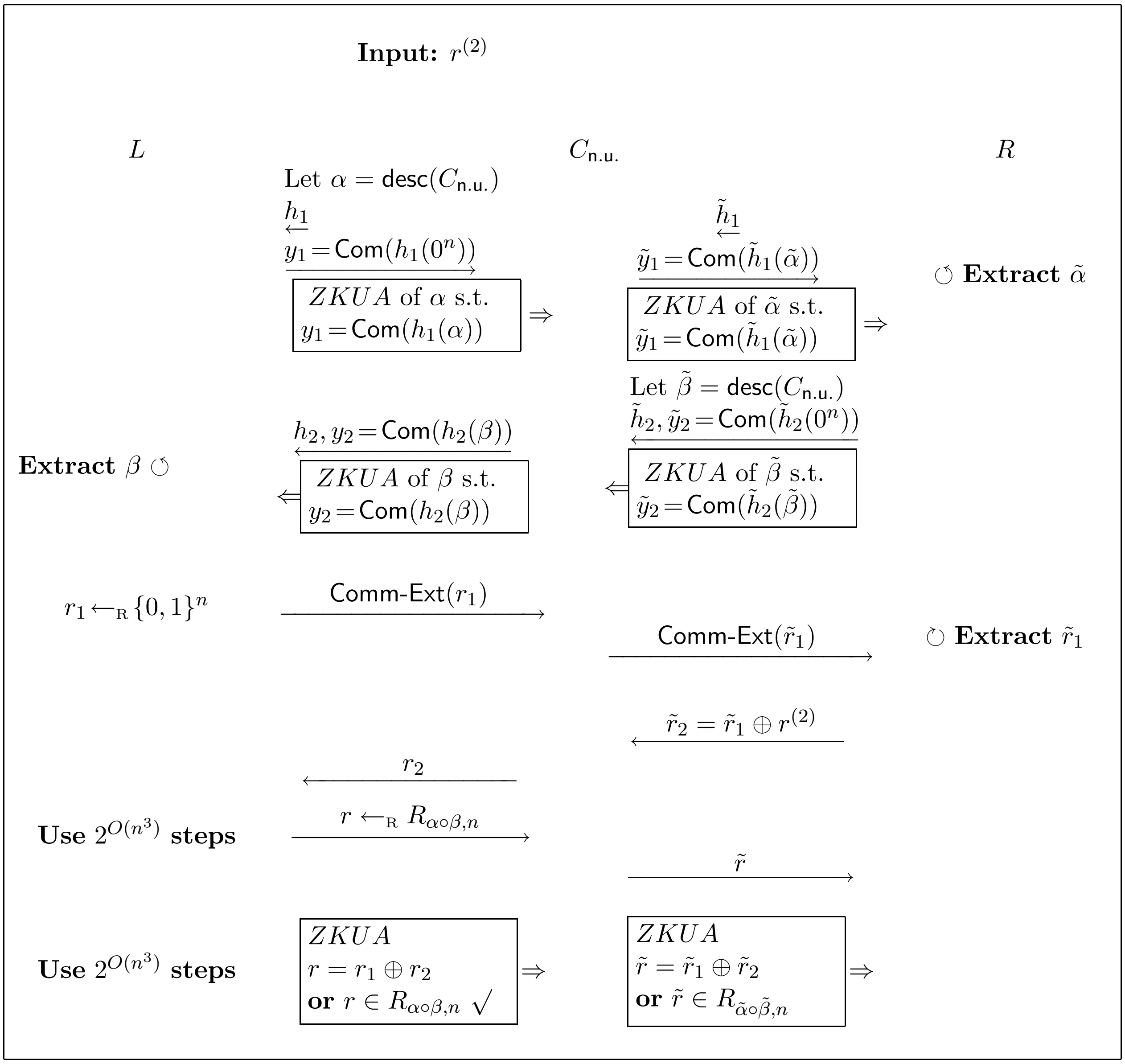


Figure 6.9: Algorithm $\widehat{C}''_{\mathsf{n.u.}}$.

3. In the corresponding steps (Steps L,R0.2.x) of the *left* session, Algorithm $\widehat{C}''_{\mathsf{n.u.}}$ uses the *extractor* of the universal argument (this may take up to $n^{O(\log n)}$ steps) and extract a string β of length up to $n^{\log n}$ and a string s such that $\mathsf{Com}(\tilde{h_2}(\beta), s) = y_2$.

4. In Step L4 of the left session, algorithm $\widehat{C}''_{\mathsf{n.u.}}$ chooses r as a random element of the set $R_{\alpha\circ\beta,n}$, using $2^{O(n^3)}$ steps. (Recall that $\widehat{C}'_{\mathsf{n.u.}}$ used in this step $r = r^{(1)}$).

5. In Steps L,R5.x of the right session, algorithm $\widehat{C}''_{\mathsf{n.u.}}$ follows the honest prover algorithm for the $ZKUA$ system, and runs in $2^{O(n^3)}$ time to prove the true statement that $r \in R_{\alpha\circ\beta,n}$. (Recall that $\widehat{C}'$ used in this step the simulator for the $ZKUA$ system.)

Now that we described Algorithm $\widehat{C}''$, we need to show that it satisfies both Properties 1 and 2 mentioned above. We start with proving that it satisfies Property 2, since this is the more interesting part.

Property 2 We need to show is that with overwhelming probability it is the case that in the transcript $\widehat{C}''_{\mathsf{n.u.}}(r^{(2)})$, $\tilde{r} \in \{r, r^{(2)}\}$. Suppose that this is not the case. This means that with non-negligible probability $\tilde{r} \notin \{r, \tilde{r}_1 \oplus \tilde{r}_2\}$ (since it $\tilde{r}_1 \oplus \tilde{r}_2 = r^{(2)}$). Firstly, note that since we are using $2^{O(n^3)} = 2^{o(n^5)}$ time, and we are doing using no rewinding in Steps L,R5.x, the soundness of the ZKUA used in these steps in the right session, ensures us that with overwhelming probability either $\tilde{r} = \tilde{r}_1 \oplus \tilde{r}_2$ or $\tilde{r} \in R_{\tilde{\alpha}\circ\tilde{\beta},n}$. This means that with non-negligible probability it holds that $\tilde{r} \neq r$ but $\tilde{r} \in R_{\tilde{\alpha'}\circ\tilde{\beta}',n}$ for some $\tilde{\alpha'}, \tilde{\beta}'$ such that $y_1 = \mathsf{Com}(h_1(\tilde{\alpha'}))$, $y_2 = \mathsf{Com}(h_2(\tilde{\beta}'))$.

We note that with overwhelming probability $\tilde{\beta}' = \tilde{\beta} = \mathsf{desc}(C_{\mathsf{n.u.}})$. Indeed, otherwise (using the extractor for the universal argument) we would have a $2^{O(n^3)}$-time algorithm for breaking the hash function. We also note that the string $\tilde{\alpha'}$ can be computed with at least $n^{-\log n}$ probability from the string $\mathsf{desc}(C_{\mathsf{n.u.}})$ in $n^{O(logn)}$-time by applying the extractor to the ZKUA of Steps L,R0.1.3.x. (This procedure will indeed get the same string α', as otherwise, using also the knowledge extractor for the universal argument of Step L,R4.x, we would have a $2^{O(n^3)}$-time algorithm that finds collisions in the hash function with non-negligible probability.)

Because the string β extracted by Algorithm $\widehat{C}''_{\mathsf{n.u.}}$ is also computed in time $n^{O(\log n)}$ from $\mathsf{desc}(C_{\mathsf{n.u.}})$ and because $\alpha = \tilde{\beta} = \mathsf{desc}(C_{\mathsf{n.u.}})$, we get that the strings $\alpha \circ \beta$ and $\tilde{\alpha'} \circ \tilde{\beta}'$ are *equivalent*. Therefore, $R_{\alpha\circ\beta,n} = R_{\tilde{\alpha'}\circ\tilde{\beta}',n}$.

We see that if we halt the algorithm $\widehat{C}'_{\mathsf{n.u.}}$ after Step L3 we get a $n^{O(\log n)}$-time algorithm with advice $\alpha \circ \beta$ that given an element $r \in R_{\alpha\circ\beta,n}$ manages to output an element $\tilde{r} \in R_{\alpha\circ\beta,n} \setminus \{r\}$. By this we obtain a contradiction to the evasiveness property of this set family.

We note that since the set $R_{\alpha\circ\beta}$ is only evasive with respect to machines that get $\alpha\circ\beta$ as advice, it is crucial that Algorithm $\widehat{C}''_{\mathsf{n.u.}}$ used the description of the code of the adversary $C_{\mathsf{n.u.}}$ as α and $\tilde{\beta}$ in Steps L,R0.1,2.x.

Property 1. We now prove that the random variable $(r^{(2)}, \widehat{C}''_{\mathsf{n.u.}}(r^{(2)}))$ (where $r^{(2)} \leftarrow_{\mathrm{R}} \{0,1\}^n$) is computationally indistinguishable by uniform algorithms from the random variable $(r^{(2)}, \widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)}))$ (where $r^{(1)}, r^{(2)} \leftarrow_{\mathrm{R}} \{0,1\}^n$). Intuitively, this follows from the secrecy of the commitment scheme, the pseudorandomness of the set $R_{\alpha\circ\beta,n}$, and the zero-knowledge property of the $ZKUA$. The actual proof, which utilizes the hybrid argument, follows.

We denote by $\mathbf{H}_0$ the random variable $(r^{(2)}, \widehat{C}'_{\mathsf{n.u.}}(r^{(1)}, r^{(2)}))$ and by $\mathbf{H}_4$ the random variable $(r^{(2)}, \widehat{C}''_{\mathsf{n.u.}}(r^{(2)}))$. We prove our claim by showing random variables $\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3$ such that for every $1 \le i \le 4$, the variables $\mathbf{H}_i$ and $\mathbf{H}_{i-1}$ are computationally indistinguishable by probabilistic polynomial-time uniform algorithms.

We now describe these intermediate random variables, and show that indeed for every $1 \le i \le 4$, $\mathbf{H}_i$ is computationally indistinguishable from $\mathbf{H}_{i-1}$. To describe the random variable $\mathbf{H}_i$, we will describe the differences between it and the variable $\mathbf{H}_{i-1}$, and then show that the two variables are indistinguishable by a class that is at least as strong as probabilistic polynomial-time uniform algorithms.

Hybrid $\mathbf{H}_1$: Commit to $C_{\mathsf{n.u.}}$'s code. The difference between $\mathbf{H}_1$ and $\mathbf{H}_0$ is that in $\mathbf{H}_1$, the honest left and right parties commit to the code of $C_{\mathsf{n.u.}}$, instead of committing to the all zeros strings. That is, in our notation, $\alpha = \tilde{\beta} = \mathsf{desc}(C_{\mathsf{n.u.}})$. The variables $\mathbf{H}_1$ and $\mathbf{H}_0$ are computationally indistinguishable by polynomial-sized circuits by the security of the commitment scheme.

Hybrid $\mathbf{H}_2$: Extract β. The variable $\mathbf{H}_2$ is distributed identically to $\mathbf{H}_1$. However, in $\mathbf{H}_2$, after running the honest verifier in Steps L,R0.2.x of the left session, we use $n^{O(\log n)}$ time to run the knowledge extractor of the universal argument. We thus β that corresponds to the values hashed and committed to by the adversary the left session.

Hybrid $\mathbf{H}_3$: Choose $r \leftarrow_{\mathrm{R}} R_{\alpha\circ\beta,n}$. The difference between $\mathbf{H}_3$ and $\mathbf{H}_2$ is that in $\mathbf{H}_3$, the Left party chooses $r \leftarrow_{\mathrm{R}} R_{\alpha\circ\beta,n}$ instead of choosing $r = r^{(1)}$. The two hybrids are indistinguishable by probabilistic $n^{O(\log n)}$-time uniform algorithms by the pseudorandomness property of the evasive set family. Indeed, note that all the elements of these two random variable can be sampled using $n^{O(\log n)}$-time and using the description of $C_{\mathsf{n.u.}}$ as an advice string. This means that a $n^{O(\log n)}$-time uniform distinguisher between $\mathbf{H}_3$ and $\mathbf{H}_2$ can be converted into a $n^{O(\log n)}$-time with advice $C_{\mathsf{n.u.}}$ machine that distinguishes a random element of $R_{\alpha\circ\beta,n}$ from the uniform distribution on $\{0,1\}^n$. This is a contradiction to the pseudorandomness property of the evasive set family because $R_{\alpha\circ\beta,n} = R_{\mathsf{desc}(C_{\mathsf{n.u.}}),n}$. (This is due to the fact that the family is nice and β is computed $n^{O(\log n)}$-time computation from $\mathsf{desc}(C_{\mathsf{n.u.}})$, and hence $\alpha \circ \beta$ is *equivalent* to $\mathsf{desc}(C_{\mathsf{n.u.}})$.)

Hybrid $\mathbf{H}_4$: Use real and not simulated proof in Steps L,R4.x. The difference between $\mathbf{H}_4$ and $\mathbf{H}_3$ is that in $\mathbf{H}_4$, the left party follows the honest prover algorithm (that takes $2^{O(n^3)}$ time) to prove the true statement that either $r = r_1 \oplus r_2$ or $r \in R_{\alpha\circ\beta,n}$. The two variables are computationally indistinguishable by the zero-knowledge property of the universal argument.

The proof is finished by observing that $\mathbf{H}_4$ is in fact the random variable $(r^{(2)}, \widehat{C}''_{\mathsf{n.u.}}(r^{(2)}))$.

□

6.4 Applications of Non-Malleable Coin-Tossing

In this section we construct a *constant-round* non-malleable commitment scheme and a *constant-round* non-malleable zero-knowledge argument system. This improves over the previously known schemes of [DDN91] that utilized a logarithmic number of rounds. The protocols of this section are only shown to be secure against man-in-the-middle adversaries that use the *synchronizing* scheduling. However, in Section 6.5 we show a generic way to transform such protocols into protocols that are secure also against *non-synchronizing* adversaries, thus finishing the proof of Theorem 6.1.5. We remark that our approach to proving Theorem 6.1.5 favors modularity and simplicity of presentation, at the expense of the resulting protocols' efficiency.

Our building blocks are our non-malleable coin-tossing protocol (which is secure in the plain model, without any setup assumption) and a non-malleable zero-knowledge argument in the *shared random string model*. This latter argument is a variant of the argument system of De Santis *et al.*[DDO+01].[29] As noted in Section 6.1.5, the proof of Theorem 6.1.5 will go as follows: First we prove that the composition (using Construction 6.1.2) of a (plain model) non-malleable coin-tossing protocol with a (shared random string model) non-malleable zero-knowledge yields a non-malleable interactive zero-knowledge argument in the plain model. Then, we show how to use such an argument to obtain also a (plain model) non-malleable commitment scheme. In the course of the proof we will define and use a stronger form of non-malleability that we call *extractability in the MIM setting*. We believe that this notion (that appears already implicitly in [DDN91] and more explicitly in [DDO+01]) is interesting in its own right.

6.4.1 Extractability in the MIM setting

Recall that the idea behind non-malleability is that an adversary in the MIM attack will not be able to utilize his interaction in one session to gain something in the other session. For example, consider the case of zero-knowledge proof systems. In such systems the left party plays the part of the *prover* and the right party plays the part of the *verifier*. We assume that in the left session the honest prover proves a statement x to the adversary, while in the right session the adversary proves a (possible related) statement $\tilde{x}$ to the honest verifier. We require that, unless $x = \tilde{x}$, if the adversary convinces the verifier then it could have done so even without interacting with the honest prover in the left session. In the non-interactive setting, [DDO+01] makes a stronger requirement. They require that if the adversary can convince the verifier of the statement $\tilde{x}$ then the adversary can in fact output a *witness* for $\tilde{x}$. What this actually means is that in a non-malleable zero-knowledge system it is possible to simulate the left session using the zero-knowledge simulator and at the same time extract a witness in the right session using the knowledge extractor.

This paradigm of "simulate from the left, extract from the right" also makes sense in other contexts. Consider *commitment schemes*, where the left party is the *sender* and the right party is the *receiver*. We would want to be able to simulate the left session, *without knowing the sender's input*,while at the same time extracting the committed value of the right session. In fact, although Dolev *et al.* [DDN91] do not make this extraction requirement part of their *definition* for non-malleable commitment scheme, they note that their *construction* for commitment-scheme has this property. As they note, this fact is important when they use their commitment scheme to construct a non-malleable zero-knowledge proof system.

We now provide a formal definition for *extractability in the MIM setting*. Intuitively, a protocol is extractable in the MIM setting if it can be "simulated from the left and extracted from the right". We assume that for any protocol a function (or relation) val is defined. This function takes a transcript and returns the "intended value" consistent with this transcript. For example, in the case of a (perfectly binding) commitment scheme, $\mathsf{val}(\sigma)$ is the unique value that is consistent with the transcript σ. In the case of a zero-knowledge system, val will be a relation rather than a function such that $y \in \mathsf{val}(\sigma)$ if y is a *witness* for the statement x proved in the transcript σ. In both cases we will let $\mathsf{val}(\sigma) = \perp$ if σ is an invalid or aborting transcript. We assume that given a transcript σ and a value y it is easy to determine whether $y \in \mathsf{val}(\sigma)$.[30]

[29] We could also have used directly the system of [DDO+01]. However, we choose to use a variant of that system that is secure under possibly weaker assumptions (does not assume the existence of dense cryptosystems). We remark that in contrast to the system of [DDO+01], our zero-knowledge system is *interactive* (although still uses only a constant number of rounds).

[30] This obviously holds in the case of zero-knowledge systems. In the case of commitment schemes we can ensure

We will assume that our protocols have a *determining message*. This is one message in the protocol that determines the value of the intended value. For example, in the case of a zero-knowledge system the determining message will be the statement x. In the case of a commitment scheme the determining message will be a message that determines the committed value uniquely. We now define a function (or relation) $\mathsf{i\text{-}value}$ that takes as input a transcript of *two concurrent sessions* τ. Let τ_L and τ_R denote the transcripts of the left and right sessions in τ. We define $\mathsf{i\text{-}value}(\tau) = \mathsf{val}(\tau_R)$ if the determining message in τ_R is *not* an exact copy of the determining message in τ_L. Otherwise, (if the determining messages are equal) we let $\mathsf{i\text{-}value}(\tau) = \perp$.

To simplify notations, we will assume that in our protocols only the left party gets an input (x, y). In commitment schemes this is always the case. In zero-knowledge systems we will assume that the left party gets (x, y) where y is a witness for x, and sends x as its first message. This allows to incorporate into the definition the ability of the adversary to choose the statement he will prove in the right session, based on information he gets in the left session. We assume that x is the *public* part of the input and y is the *secret* part of the input (e.g., in the case of commitment schemes x is empty). We will also assume that the adversary does not get an input, as this can be taken care of by non-uniformity.

We can now give a formal definition for extractability in the MIM setting:

Definition 6.4.1. Let $\Pi = (L, R)$ be a two party protocol with an intended value function $\mathsf{i\text{-}value}$ defined as above. We say that Π is *extractable in the MIM setting* if for any polynomial-sized MIM adversary C there exists a (standalone) simulator with 2-outputs $\widehat{C}$ such that for any inputs (x, y) to the honest left party:

1. If $(\tau', y') \leftarrow \widehat{C}(x)$ then with overwhelming probability either $\mathsf{i\text{-}value}(\tau') = \perp$ or $y' \in \mathsf{i\text{-}value}(\tau')$.

2. Let τ denote C's view when interacting with $L(x, y)$ and R, and let τ' denote the first output of $\widehat{C}(x)$. Then τ and τ' are computationally indistinguishable.

If $\mathsf{i\text{-}value}$ is a function (and not a relation) then we can make the following stronger requirement in addition to Items 1 and 2: We say that an extractable scheme Π is *strongly extractable* (in the MIM setting) if the distribution $(\tau, \mathsf{i\text{-}value}(\tau))$ conditioned on $\mathsf{i\text{-}value}(\tau) \neq \perp$ is computationally indistinguishable from the distribution (τ', y') conditioned on $\mathsf{i\text{-}value}(\tau') \neq \perp$, where τ is C's view when interacting with $L(x, y)$ and R and $(\tau', y') \leftarrow \widehat{C}(x)$. (Note that the fact that a protocol is extractable does not imply immediately that it is strongly extractable: it may be the case that τ is indistinguishable from τ' but $\mathsf{i\text{-}value}(\tau)$ is in fact distinguishable from $\mathsf{i\text{-}value}(\tau')$ since $\mathsf{i\text{-}value}$ is not an efficiently computable function.) Note that in both variants, we allow the simulator to output an arbitrary value as its second output, if its first output τ' satisfies that $\mathsf{i\text{-}value}(\tau') = \perp$.

Since we only deal with the MIM setting from now on we will use the names *extractable* and *strongly extractable* and drop the qualifier "in the MIM setting". We say that Π is extractable (resp. strongly extractable) *with respect to non-synchronizing adversaries* if the extractibility condition (resp. strong extractibility condition) holds only against MIM adversaries C that use the *synchronizing* scheduling.

Relationship with non-malleability definition. As mentioned above, the condition of extractiblity is stronger than the notion of non-malleability as defined originally in [DDN91]. Dolev *et*

---

this holds by appending $\mathsf{val}(\sigma)$ with some auxiliary information *al.*defined a non-malleable protocol as a protocol where an MIM adversary cannot succeed in causing a non-trivial[31] relation between the left party's input and the intended value of the right session, more than can a simulator that only receives the public information. An extractable protocol is non-malleable because its second output will hit the relation with probability at most negligibly smaller than the adversary's. We note that latter works (such as [DDO+01] in the common reference string model) chose to use the condition of extractiblity as the *definition* of non-malleability, since this stronger condition is often needed in applications.

6.4.2 Constructing Extractable Protocols

Our approach to constructing an extractable zero-knowledge scheme and a strongly extractable commitment scheme is the following:

1. Reduce the problem to constructing only an extractable zero-knowledge scheme.
2. Show that our non-malleable coin-tossing protocol allows us to reduce the problem to the problem of constructing such a scheme in the *modified shared random string model*.
3. Give a simple construction of an extractable zero-knowledge scheme in the modified shared random string model
4. Show (in Section 6.5) that for both commitment and zero-knowledge schemes, one can transform an extractable protocol secure against synchronizing adversaries into a protocol secure also against non-synchronizing adversaries.

The following lemma handles Step 1 of this outline. That is, it says that in order to obtain both extractable commitments and zero-knowledge schemes, it is enough to construct the latter. We state and prove the lemma only for the case of syncrhonizing adversaries since this is the version we need in this section.

Lemma 6.4.2. *If there exists a standard (non-interactive) perfectly binding commitment scheme* Com *and a zero-knowledge argument that is extractable w.r.t. synchronizing adversaries then there exists a commitment scheme that is strongly extractable w.r.t. synchronizing adversaries.*

Proof Sketch: To commit to a value y, run the following protocol:

Step L1 Left sends $\mathsf{Com}(y)$.

Step L,R2.x Left proves to right knowledge of the committed value using an extractable zero-knowledge argument.

For a synchronizing adversary C, denote by $\tilde{\alpha} = \mathsf{Com}(\tilde{y})$ the random variable representing C's first message in the right session. Suppose that use the simulator of the zero-knowledge scheme to simulate the proof of Steps L,R2.x (but still commit to y in Step L1). Since C is synchronizing, the distribution of $\tilde{\alpha}$ is unchanged. Yet, now we are able to extract $\tilde{y}$.

Now, suppose that we use $\mathsf{Com}(0^n)$ instead of $\mathsf{Com}(y)$ in the first step. No noticeable change should occur in the joint distribution of the transcript and $\tilde{y}$, since otherwise we would contradict the semantic security of the commitment scheme Com. Therefore our simulator will use a commitment to 0^n in the first step, and the simulator for the zero-knowledge argument in the second step.

□

[31] In this context, we'll say that a relation R is non-trivial if $(x, x) \notin R$ for every x. This condition is required to rule out the trivial adversary that uses the relaying strategy in order to hit the relation.

Note that the commitment scheme constructed has the property that the first message sent is from the sender to the receiver and that this message completely determines the committed value. We say that such a scheme has a *determining first message*.

6.4.3 Extractable Schemes in the Modified Shared Random String Model

We now define extractable and strongly extractable protocols in the *modified shared random string model*. The only difference is that we assume that $(r^{(1)}, aux^{(1)})$ and $(r^{(2)}, aux^{(2)})$ are generated independently by a generator algorithm (where r is random or pseudorandom), and then C chooses whether $r^{(1)}$ or $r^{(2)}$ will be used in the right session. The simulator will get $aux^{(1)}, aux^{(2)}$ as an additional input. We note that the resulting definition for extractable zero-knowledge schemes is almost identical to the definition of (single theorem) non-malleable NIZK [DDO+01]. The following theorem says that a non-malleable coin-tossing algorithm can indeed be used to "compile" a protocol secure in the modified shared random string model to a protocol secure in the plain model. Again, we state and prove the theorem only for the case of synchronizing adversaries.

Theorem 6.4.3. *Let Π be a non-malleable coin-tossing protocol and let Π_{Ref} be a protocol with a reference string that is extractable (w.r.t. synchronizing adversaries) in the modified shared random string model. Let $\Pi \circ \Pi_{\mathsf{Ref}}$ be the composition of Π and Π_{Ref} using Construction 6.1.2, then Π_{Ref} is extractable (w.r.t. synchronizing adversaries) in the plain model.*

Proof. Let C be an adversary for the combined protocol $\Pi \circ \Pi_{\mathsf{Ref}}$. Suppose that C is synchronizing. Then we can separate C into C_1, C_2 where C_1 is the adversary's strategy for the first phase (coin-tossing) and C_2 is the adversary's strategy for the second phase (running Π). We will pick $(r^{(1)}, aux^{(1)})$ and $(r^{(2)}, aux^{(2)})$ using the generator and then simulate C_1 using the simulator of the coin-tossing protocol ,where we give it $r^{(1)}, r^{(2)}$ as input. Let s denote the simulated output. Now, we can look at C_2 with the state s hardwired into it as an adversary for the extractable protocol Π in the modified shared random string model and we can simulate it using the simulator Π and obtain (τ', y') such that $y' \in \mathsf{i\text{-}value}(\tau')$. Yet τ' must be computationally indistinguishable from the output of C in an execution of $\Pi \circ \Pi_{\mathsf{Ref}}$, and so we're done.

□

6.4.4 An Extractable Zero-Knowledge Argument in the Modified Shared Random String Model

We now show a simple protocol for an extractable zero-knowledge argument in the modified shared random string.

Outline of the zero-knowledge argument. Our zero-knowledge argument is specified in Protocol 6.4.4. It has the following form: to prove that $x \in L$ when given a reference string r, the prover treats r as a verification key of some signature scheme, and proves (using a standard constant-round interactive zero-knowledge proof of knowledge)[32] that either $x \in L$ or that it knows a signature on x w.r.t. this verification key.

We see that our protocol requires a signature scheme that has a uniformly distributed public key. Fortunately, we only need a *one-time length restricted* signature scheme and so we can use the simple and well-known construction due to Lamport [Lam79], instantiating it with a one-way *permutation*.[33]

[32] Actually, it is enough to use a witness indistinguishable argument of knowledge.

[33] That is, the public key is a list y_i^σ where $i \in [n]$ and $\sigma \in \{0,1\}$, and to sign a message $m = m_1, \ldots, m_n \in \{0,1\}^n$,

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$"), $r \in \{0,1\}^{l(n)}$: the reference string **Prover's auxiliary input:** w (a witness that $x \in L$)	$w \downarrow$ $x, r \downarrow$ $\boxed{P}$ $\boxed{V}$
Steps P,V1.x (WI Proof): Prover proves to verifier using its input w via a witness-indistinguishable (WI) proof/argument of knowledge that either $x \in L$ or it knows a signature on x w.r.t. r where r is considered to be the public-key of a fixed one-time length-n message-size signature scheme (G, S, V). Verifier accepts if proof is completed successfully.	$w \downarrow$ $x, r \downarrow$ *WI-proof* $x \in L$ **or** know σ s.t. $V_r(x, \sigma) = 1$ $\downarrow$ 0/1

Protocol 6.4.4. An extractable zero-knowledge argument in the modified shared random string model

The theorem that we need to prove is the following:

Theorem 6.4.5. *Let (G, S, V) be a one-time length-n message signature scheme whose public key is a distributed uniformly in $\{0,1\}^{l(n)}$. Then, the instantiation of Protocol 6.4.4 with (G, S, V) is an extractable zero-knowledge argument in the modified shared random string model.*

Proof. Let C be an MIM adversary for Protocol 6.4.4. We construct a simulator $\widehat{C}$ for C in the following way:

1. Let $r^{(1)} = G(aux^{(1)})$ and $r^{(2)} = G(aux^{(2)})$ where G is the key generation algorithm of the signature scheme (G, S, V) and $aux^{(1)}, aux^{(2)}$ are two independently chosen random coin-tosses for G.

2. The simulator invokes $C(r^{(1)}, r^{(2)})$ and obtains a number $b \in \{1, 2\}$ such that if $b = 1$ then $r^{(1)}$ will be used as the reference string in the right session, and if $b = 2$ then $r^{(2)}$ will be used as this string.

3. The simulator computes a signature σ on x with respect to $r^{(1)}$ using $aux^{(1)}$. Using this signature σ, it construct a left-side strategy L_σ for the left side that uses the honest prover algorithm of the WI system to prove the true statement that either $x \in L$ or it knows a signature σ on x with respect to $r^{(1)}$.

4. The simulator simulates an interaction between L_σ, the adversary C, and the right side R. Let $\tau = (\tau_L, \tau_R)$ be the resulting view of the adversary, where τ_L (resp. τ_R) denotes the view of the adversary in the left (resp. right) session. We denote the statement proven by the adversary in the right session by $\tilde{x}$.

5. If in the view τ_R the verifier *accepted* the proof given by the adversary C, then the simulator will treat L_σ and C as a combined prover algorithm for the WI system and use the extractor of the WI system to either a witness $\tilde{y}$ that $\tilde{x} \in L$ or a signature $\tilde{\sigma}$ on $\tilde{x}$ w.r.t. $r^{(2)}$.[34]

one sends $x_i^{m_i} = f^{-1}(y_i^{m_i})$ for all $i \in [n]$, where $f(\cdot)$ is the one-way permutation.

[34] One needs to introduce a time-out mechanism in order to make sure this procedure runs in expected polynomial-time, in a similar way to [Lin01]. In fact, one can use directly a *witness-extended emulator* [BL02, Lin01] instead of a knowledge extractor. However, it seems that describing the simulator that uses the knowledge extractor is slightly simpler.

6. The simulator outputs the view τ. In addition, if it obtained in the previous step a witness $\tilde{y}$ that $\tilde{x} \in L$ then it outputs this witness. Otherwise, its second output is $\perp$

By the WI condition, the simulator's first output is indeed indistinguishable from the view of C in a real interaction. Also, by the proof of knowledge condition, if the proof in the right session passes verification, then the simulator will indeed obtain either a witness $\tilde{y}$ for $\tilde{x}$ or a signature $\tilde{\sigma}$ on $\tilde{x}$ with respect to $r^{(2)}$. Thus, all that is left is to show that if $\tilde{x} \neq x$, then the second case (i.e., that the simulator obtains a signature) happens with negligible probability. To show this, one needs to observe that in Steps 4 and 5, the only secret information that the simulator uses is σ, which is a signature on x w.r.t. $r^{(1)}$. In particular, the simulator does *not* use either $aux^{(1)}$ or $aux^{(2)}$ in these steps. Therefore, if it managed to obtain a signature for any message $\tilde{x} \neq x$ w.r.t. $r^{(1)}$ in these steps that would contradict the existential unforgeability of the signature scheme (G, S, V) after seeing a single signature. If it managed to obtain a signature on *any* message (even on x) w.r.t. $r^{(2)}$ that would mean that it managed to forge even without seeing a single signature. □

6.5 Handling Non-Synchronizing Adversaries

Throughout this paper, we have always concentrated on the case of adversaries that use the *synchronizing* scheduling. In this section, we justify this, by showing a way to transform protocols that are secure against synchronizing adversaries, into protocols that are secure against adversaries that may also use a *non-synchronizing* scheduling. (See Section 6.1.2 and Figure 6.3 for the definition of the synchronizing and non-synchronizing scheduling.)

Outline of our approach. The underlying observation behind the transformation of this section is that in some sense the synchronizing scheduling is the hardest schedule to deal with. To illustrate this, consider a malleable zero-knowledge proof of knowledge (P, V). We know that (P, V) is malleable, but it is still instructive to try to prove that it is non-malleable and see exactly in what place the proof fails. Suppose that we are trying to prove that (P, V) is extractable (in the sense of the previous section). The natural approach for a simulator would be to try to invoke the stand-alone simulator of (P, V) to simulate the left session, and the stand-alone extractor of (P, V) to extract a witness from the right session. The problem with this approach is that when simulating the left-session, one needs to combine both the adversary C and the right party R and to treat them as one single verifier algorithm. The problem is that when simulating a proof for this combined verifier, the simulator needs to *rewind* it in the process. This means that the simulator rewinds not only the adversary, but also the right party. As a consequence, when will fail when we try to run the knowledge extractor to extract the witness in a right session. Technically, the reason is that to run the extractor one needs to treat the simulator and the adversary as one combined prover algorithm for the proof system of the right session. However, because the simulator has the power to rewind the right party, it is not a "legitimate" interactive prover, and therefore the knowledge extractor cannot be applied to it.

We saw that we got intro trouble when we tried to simulate the left session and rewind the adversary, but needed to rewind the right party along with it. However, this is not necessarily the case if the adversary uses a *non-synchronizing* strategy. As can be seen in Figure 6.5, in this case there will be at least one point in the left session in which the adversary will return a message immediately to the left party, without any interaction with the right party (in Figure 6.5, this point is marked with a $\circlearrowleft$). If it happens to be the point at which the simulator needs to rewind, then the simulator can rewind "safely" without having to rewind also the right party at the same time.

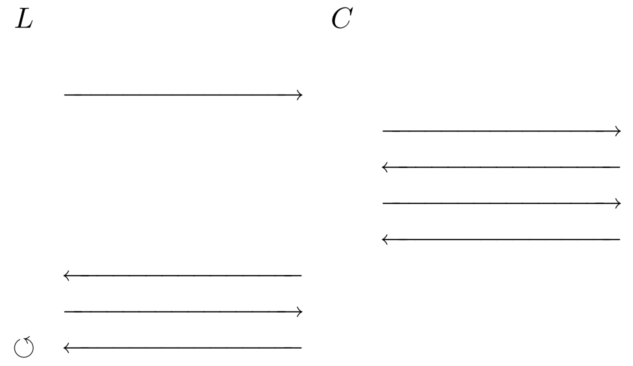


Figure 6.10: A non-syncrhonizing adversary: It is "safe" to rewind at the point marked by ↺.

The question that remains is how to make sure that the point at which the adversary deviates from the synchronizing scheduling will be the same point in which the simulator needs to rewind. To do so, we follow the Richardson-Kilian [RK99] approach of *multiple rewinding opportunities* (See also [GL00, PRS92]).[35] That is, we will make essentially every point in the protocol a possible rewinding point. Therefore, whenever the adversary deviates from the synchronizing scheduling, we'll be able to use this to simulate it, and in fact have a simpler simulation, compared to the simulators for synchronizing adversaries of the previous sections.

We now turn to the formal statement and proof of the theorem. We will start with the case of commitment schemes, and then describe how the proof can be modified for the case of zero-knowledge argument.

Theorem 6.5.1. *Suppose that there exists a constant-round strongly extractable commitment scheme with respect to synchronizing adversaries and that there exist perfectly hiding commitment schemes. Then, there exists a constant-round strongly extractable commitment scheme (with respect to adversaries that use arbitrary scheduling).*

6.5.1 Proof of Theorem 6.5.1

Let $\Pi_1 = (L, R)$ be a commitment scheme satisfying the conditions of the theorem. (That is, it is strongly extractable with respect to synchronizing adversaries.) We will prove Theorem 6.5.1 in the following way:

1. First, we will show a relatively simple construction of a commitment scheme Π_2 that is strongly extractable with respect to *non-synchronizing* adversaries. That is, for every MIM adversary C, there exist a 2-output simulator for C as long as C utilizes a *non-synchronizing* scheduling (but may fail if the adversary uses a synhcronizing scheduling).[36] The existence of such a scheme Π_2 lies behind our intuition that the synchronizing scheduling is the harder case to deal with. The construction of Π_2 will follow the Richardson-Kilian [RK99] paradigm of multiple rewinding points.

2. Secondly, we will construct a protocol Π which is a combination of Π_1 and Π_2. Roughly speaking, when committing to a string y, the first message of Π will be a two commitments

[35] We note that this approach was already used implicitly in a context similar to ours by the earlier paper [DDN91].

[36] Strictly speaking, the simulator will be slightly weaker than this, since we will allow it to fail also if the adversary utilizes an "almost synchronizing" scheduling, where this term will be defined below.

to two random strings y_1 and y_2 such that $y_1 \oplus y_2 = y$. Then, we'll run Π_1 (committing to y_1) and Π_2 (committing to y_2) in parallel. That is, each message of Π will consist of one message from Π_1 and one message of Π_2.

3. We will then show that Π is a strongly extractable commitment scheme with respect to adversaries that utilize *any* scheduling strategy. Loosely speaking, for every adversary C, if C uses a synchronizing scheduling then we'll simulate it using the simulator of Π_1, and if C uses a non-synchronizing scheduling, then we'll simulate it using the simulator of Π_2.

Residual strategies. Before continuing with the proof, we recall the notion of *residual strategies* that will be useful for us later on. Recall that an interactive algorithm A is an algorithm that computes a *next-message function*. That is, given a random-tape r, and a list of messages $m_1, \ldots, m_i$, Algorithm A computes the next message m that it would send in an execution in which it has r as the random tape and received from the other party the messages $m_1, \ldots, m_i$. We use the notation $m = A(m_1, \ldots, m_i; r)$. If A is some interactive strategy and $v = \langle r, m_1, \ldots, m_j \rangle$ is a *partial view* of A, then the *residual strategy* of A with v fixed, is the next-message function that is obtained by "hardwiring" into A the view v. That is, this is the function that given a list of messages $m_{j+1}, \ldots, m_i$ (where $i \geq j+1$) outputs $A(m_1, \ldots, m_j, m_{j+1}, \ldots, m_i; r)$. In the following sections, we will often consider an execution of an interactive algorithm A up to some point in a protocol, and then consider the residual algorithm A, which has the view up until that point fixed.

Construction of the scheme Π_2.

We now describe the construction of a commitment scheme Π_2 that will be secure with respect to *non-synchronizing* adversaries.

RK-iterations. To describe the scheme Π_2, we introduce a notion from [RK99], which we call an *RK-iteration*. An *RK-iteration* is a three round subprotocol α, β, γ, where the first and last messages are from the right party to the left party. Loosely speaking, we will define a condition under the left party is said to *win* an RK-iteration. The property that we require from an RK-iteration is that the probability that the left party wins in a normal interaction is negligible, *but* if the left party has the power to rewind the right party at the point β (i.e., get an additional reply γ' to some query $\beta' \neq \beta$) then it will be able to win with probability 1. To make this more concrete, we use the following implementation for an RK-iteration:

Let (G, S, V) denote a one-time length-restricted signature scheme. (Note that under the conditions of the theorem there exist one-way functions, and therefore there also exist such signature schemes [Lam79].) We use the following RK-iteration protocol:

Step R1 (Send VK): Right party runs $G(1^n)$ and obtains a verification and signing key-pair $\langle \alpha = VK, SK \rangle = \langle G_1(1^n), G_2(1^n) \rangle$. It sends the verification key, which we denote by α, to the left party.	$\xleftarrow{\alpha = G_1(1^n)}$
Step L2 (Send message): Left party chooses a message $\beta \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends it to the right party.	$\xrightarrow{\beta \leftarrow_{\mathrm{R}} \{0,1\}^n}$
Step R3 (Send signature): Right party computes a signature γ on the message β using the signing key, and sends it to the left party. If the right party does not send a valid signature at this step then we consider it as aborting the protocol.	$\xleftarrow{\gamma = S_{SK}(\beta)}$

Public input: 1^n (the plaintext string that will be committed to is of length n) **Left's (Sender's) auxiliary input:** $y \in \{0,1\}^n$ (plaintext to be comitted to)	$\underset{\downarrow}{w}$ $\underset{\downarrow}{x, r}$ $\boxed{L}$ $\boxed{R}$
Step L1 (Standard commitment): Left sends to right a commitment to y using a standard (possibly malleable) non-interactive commitment Com	$\xrightarrow{\mathsf{Com}(y)}$
Steps L,R$2 \cdots 2c + 2$ (RK-iterations): Left and right perform c RK-iterations. Note that we can combine the first step of each iteration with the last step of the previous one.	Repeat c times: $\xleftarrow{\alpha}$ $\xrightarrow{\beta}$ $\xleftarrow{\gamma}$
Steps L,R$2c + 3 \cdots 2c + 5$ (WI Proof): Left proves to right using a perfect-WI argument of knowledge that it either knows the value y committed to in Step L1 or that it won one of the RK-iterations (i.e., for some RK-iteration $\langle \alpha, \beta, \gamma \rangle$ it knows a signature δ w.r.t. α on some message $\beta' \neq \beta$). Right accepts if proof is completed successfully.	Perfect $WIPOK$ know y **or** $\exists$ RK-iteration $\langle \alpha, \beta, \gamma \rangle$ s.t. know δ, β' s.t. $\beta' \neq \beta$ and $V_\alpha(\beta', \delta) = 1$

Protocol 6.5.2. Π_2: A commitment scheme strongly extractable with respect to *non-synchronizing* adversaries.

We say that the left party *wins* the iteration if it knows (i.e., can output on an auxiliary tape) a valid signature with respect to the verification key α on some message $\beta' \neq \beta$. Clearly, the probability that an efficient left party wins an RK-iteration is negligible.

Description of the scheme Π_2. Let c denote the number of left party messages in the scheme Π_1.[37] The scheme Π_2 consists of the following: to commit to a value y, the left party sends a standard non-interactive commitment to y, and then the left and right parties perform c RK-iterations. After this, the left party proves using a perfectly witness-indistinguishable proof of knowledge that either it knows the value it committed to, or that it won one of the RK-iterations. For completeness, we provide a more formal description in Protocol 6.5.2.

Notation. Consider the left-party messages of the scheme Π_2 (Protocol 6.5.2). Note that c of these messages consist of a second message β of some RK-iteration. We call these messages the *core* of the scheme Π_2. We say that an MIM adversary C for Π_2 uses an *almost synchronizing* scheduling if it's scheduling is synchronized with respect to all messages in the core. The scheduling of the other messages may be unsynchronized. More formally, let us number all the messages of the protocol from 1 to $2c + 5$ (and so the left-party messages get odd numbers and the right-party

[37] Since we can always add "dummy" messages to Π_1, we assume without loss of generality that the left party sends the first and last message in Π_1 and so the number of rounds in Π_1 is $2c - 1$.

messages get even numbers). We see that the set of core messages consists of all odd numbers between 3 and $2c+1$. We say that a scheduling for Π_2 is *almost synchronizing* if for every i in the core, the i^{th} (left-party) message is sent in the *right* session before the $i+1^{th}$ (right-party) message is sent in the *left* session. We note that we only refer to the scheduling of messages sent by the adversary since we can always assume without loss of generality that the honest parties send their messages immediately after they receive the previous message from the adversary. If a schedule is *not* almost-synchronizing then there exists a smallest i in the core for which the $i+1^{th}$ (right-party) message is sent in the left session *before* the i^{th} (left-party) message is sent in the right session. We call this i the *first unsynchronized core message* and we denote the RK-iteration that this message is part of as the *first unsynchronized RK-iteration.*

We have the following proposition:

Proposition 6.5.3. *The Scheme Π_2 is strongly extractable with respect to any adversary that uses a scheduling that is* not *almost-synchronizing.*

Proof. Let C be an adversary for Π_2 that utilizes a scheduling that is *not* almost-synchronizing. To show that Π_2 is strongly extractable we need to exhibit a simulator S for C. For every $y \in \{0,1\}^n$, the simulator S needs to simulate both the view of the adversary and the distribution of the value committed to by the adversary in the right session, when the adversary interacts with $L(y)$ and R (where $L(y)$ denotes the strategy of the left party when committing to input y). However, the simulator does *not* get y as an input, but only 1^n. Initially, we will construct a simulator S' that gets as input a string $e = \mathsf{Com}(y)$ as an input. Later, we will show how to get rid of this assumption, and obtain a simulator S satisfying the desired properties.

This simulator S' will work in the following way. The simulator will run internally an execution between the adversary C and the parties L and R. Note that the simulator can send $e = \mathsf{Com}(y)$ as its first message, and does not need to know y until the WI proof stage of the execution. Therefore, it can simulate perfectly the left party's strategy until that stage.

Since the adversary is not using an almost synchronizing strategy, at some point in the execution there will be a first unsyncrhonized core message i, as defined above. This is some point in which the left party sent its message of step i (which is the second step β of some RK-iteration) and obtained from the adversary a response γ immediately, without any interaction between the adversary and the right party taking place between the left party's message and the adversary's response. At this point, the simulator will run the following experiment: it will rewind the adversary to the point just before the i^{th} message was sent by the left party, and query the adversary with a new random message β'. The simulator will repeat this until the adversary again responds immediately to some β' with some answer γ'. (The overall number of steps will be expected polynomial number of times, since with probability p the simulator runs in $\frac{1}{p}\text{poly}(n)$ steps.)

The simulator will then record β' and γ', and continue to run the execution. However, it will now use a different residual strategy for the left party. Instead of using L, which is the honest left strategy, the simulator will use the strategy L' which when it gets to the WI proof stage, does *not* use the knowledge of the committed string, but rather uses its knowledge of β' and γ' to prove that it "won" some RK iteration. The simulator will continue the execution, and the view τ of the adversary in this execution will be the simulator's first output. Note that this view τ is distributed identically to the view of an adversary in a real execution because of the perfect WI condition.

Recall that the simulator needs also to output the message committed in the right session of τ, the simulator will rewind the parties L', R and the adversary C to the point in τ just before the adversary makes its proof of knowledge to the right party. It will then combine the adversary with the residual left strategy L' to obtain a prover algorithm for this system. The simulator then uses this prover as input to the knowledge extractor of the system and obtains either the message committed to by the adversary or a signature w.r.t. a verification key of some RK-iteration in the right session, which is on a message *different* from the message β of that iteration. However, because no rewinding of the right party at any of the RK iterations is done, the probability of the latter event is negligible (or otherwise we would have a forging algorithm for the signature scheme) and therefore with very high probability the simulator obtains the committed string that it needs as its second output.

Note that the distribution on the pair of the simulator's outputs is statistically indistinguishable from the distribution of the adversary's view and adversary's committed value in a real execution. Therefore, this simulator S' satisfies the strong extractability condition. To obtain a simulator S that does *not* get a $e = \mathsf{Com}(y)$ as an additional input, we simply have $S(1^n) = S'(\mathsf{Com}(0^n))$. By the indistinguishability of the commitment scheme Com, the pair output by S will be computationally indistinguishable from the pair output by S'. □

Remark 6.5.4. Note that the proof of Proposition 6.5.3 actually shows a somewhat stronger condition than the statement. Not only does the simulator computes a view and witness pair (τ, y), but it also computes the left strategy L' that when executed with the adversary yields the view τ. We will later use this property of the simulator.

Construction and analysis of the scheme Π.

Now that we have a scheme Π_1 that is secure against adversaries that utilize the *synchronizing* scheduling, and a scheme Π_2 that is secure against adversaries that utilize a *non-synchronizing* scheduling, we are ready to construct our scheme Π which will be secure against adversaries that use arbitrary scheduling.

Properties of the Scheme Π_1. We note that we may assume that the scheme Π_1 has a *determining first message*,[38] as one can convert a scheme without this property into a scheme with this property by adding an initial message in which the left party sends a standard commitment to the right party. We also note that, as an extractable commitment scheme, the scheme Π_1 satisfies a standalone extraction property (or equivalently, it is a commit-with-extract scheme). That is, if we consider a *standalone* (not man-in-the-middle) execution of the scheme Π_1, where the left party is possibly cheating, then we can both simulate such an execution and obtain the string committed to by the left party. This is because one can regard the cheating left party as a degenerate man-in-the-middle adversary that does not use the messages it sees in the left session in its interaction in the right session. Such a degenerate adversary can without loss of generality use any scheduling (and in particular the synchronizing scheduling) and therefore we can use the simulator for Π_1 to obtain a simulated transcript of the right session along with the value committed to in this transcript.

The scheme Π. As outlined above, the combined scheme Π will be as follows:

1. The left and right parties run both protocols Π_1 and Π_2 in parallel to commit to y_1 and y_2 respectively. Note that Π_2 has more rounds than Π_1. We align the rounds of Π_1 to the *core* rounds of Π_2. That is, along with each of the c messages of the form β in rounds $3 \cdots 2c+1$ of Π_2, the left party sends a message that belongs to Π_1.

[38] Recall that this means that first message in the protocol determines the possible value for the commitment.

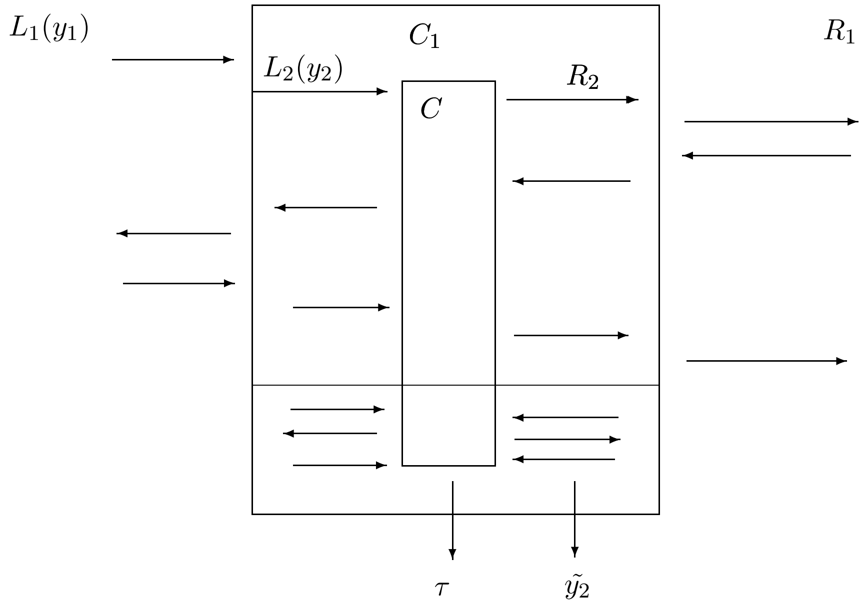


Figure 6.11: Converting adversary C to an adversary C_1 for the protocol Π_1.

We claim that Π is a strongly extractable commitment scheme. Indeed, let C be an adversary for the scheme Π, we now describe a simulator S for C:

Sampling The simulator executes internally an execution of the protocol Π with C as a man-in-the-middle, where the left party commits to the value 0^n. The simulator then checks whether the scheduling of the adversary was almost synchronizing or not. That is, whether all messages in the core of Π_2 (and thus *all* messages of Π_1) are scheduled in a synchronized way.

The almost-synchronized case Suppose that in the sampled execution the adversary C used an almost synchronizing scheduling. In this case the simulator will use the simulator for Π_1. To do so, the simulator converts the adversary C into an adversary C_1 for the protocol Π_1. See Figure 6.11 for a schematic description of the adversary C_1. The adversary C_1 chooses a string $y_2 \leftarrow_{\mathrm{R}} \{0,1\}^n$, and runs an internal copy of C and an internal copy of the left strategy (committing to y_2) and right strategy for the protocol Π_2. When the adversary C_1 plays man-in-the-middle in the execution of Π_1 it forwards the messages it receives to its internal copy of C. Recall that since C is an adversary for Π, it also expects messages that belong to Π_2. To obtain these messages, the adversary C_1 uses its internal copy of the left and right strategy for Π_2. The adversary C_1 will be a *synchronizing* adversary for the protocol Π_1. This means that if its internal copy of C tries to schedule a message of Π_1 (or equivalently, a core message of Π) in a non-synchronized way then adversary C_1 will abort and output a special fail symbol. Otherwise, it will continue until the end of the protocol Π_1. Note that after the protocol Π_1 is finished, C_1's internal copy of adversary C will still not finish the last part of protocol Π (which consists of the proof of knowledge phase of Protocol Π_2). Adversary C_1 will simulate this part internally and use the knowledge extractor to obtain the string $\tilde{y}_2$ that

C committed to in the right session.[39] The output of adversary C_1 is the view of its internal copy of the adversary C, along with the string $\tilde{y}_2$ committed by this internal adversary.

Note that if $y_1 = y \oplus y_2$ then (conditioned on being different from fail) the output of the adversary C_1 is identical to the view of the adversary C in an execution of Π. The simulator S now runs simulates C_1 using the simulator for Π_1 to obtain a pair $(\tau, \tilde{y}_2, \tilde{y}_1)$ where $\tau, \tilde{y}_2$ is computationally indistinguishable from C_1's output[40] and $\tilde{y}_1$ is the value committed by the adversary in the right session of the execution corresponding to τ. (If the simulator outputs a pair where $\tau =$ fail then we repeat the experiment until we get a non-fail output.) Therefore, the pair $(\tau, \tilde{y}_1 oplus \tilde{y}_2)$ is computationally indistinguishable from the view and committed value of C in a real execution of Protocol Π.

The non-synchronizing case Suppose that in the sampled execution the adversary C did *not* use an almost-synchronized scheduling. In this case, we will use the simulator of Π_2 in an analogous way to the way we used the simulator of Π_1 in the previous step. That is, we will construct an adversary C_2 for the protocol Π_2 that has an internal copy of C inside it. The adversary C_2 will forward to C all the messages that it receives from the parties L_2 and R_2 in the protocol Π_1. In addition, it will execute internal copies of the honest left and right parties L_1 and R_1 of the protocol Π_1 (where it will provide L_1 with y_1 that is chosen at random in $\{0,1\}^n$). It will use these internal copies to supply the internal adversary C with the messages that it expects that come from the protocol Π_1. At the end of the protocol, the adversary C_2 outputs the view of its internal copy of C. As in a previous case, the output of C_2 after interacting with $L_2(y_2 \oplus y)$ and R_2 is identically distributed to the view of C after interacting with $L(y)$ and R. Therefore, if we simulate C_2, we obtain a pair of a transcript τ and the value $\tilde{y}_2$ committed to in the right session of Π_2 that indistinguishable from a transcript/value pair in a real execution of Π. The only thing that is missing is to obtain the value $\tilde{y}_1$ corresponding to this execution. However, this can be done using the fact that the simulator for protocol Π_2 constructed in the proof of Proposition 6.5.3 actually supplied us with a residual strategy for the left party L_2 that yields the transcript τ. Therefore, we can combine this left party L_2 with the adversary C to obtain a *standalone* adversary for the commitment scheme Π_1. Then, we can use the standalone extraction property of Π_1 to obtain the desired value $\tilde{y}_1$.

We see that both in the synchronized and unsynchronized case, the simulator S outputs a transcript/value pair that is indistinguishable from the transcript/value pair of a real execution of the adversary C. Intuitively, if we let p be the probability that C uses an almost synchronized scheduling, then the expected running time of the simulator will be $p\frac{1}{p}\mathrm{poly}(n) + (1-p)\frac{1}{(1-p)}\mathrm{poly}(n)$ which is equal to some polynomial $q(n)$ (Because, the simulator S will need to invoke the simulators for Π_1 and Π_2 a number of times that depends on $\frac{1}{p}$ until it gets another sample with the same property). However, because the probabilities in the simulation may have a negligible difference from the sampled probabilities, this is actually not the case, and we need to use the standard trick of [GK96] (see also [Lin01]) and introduce a timeout mechanism. That is, in the sampling stage, the simulator will actually sample many times an execution until it has an estimate $\tilde{p}$ for p that is within a factor 2 of p with probability $1 - 2^{-n^2}$ (this will mean that with probability p we sample

[39] Actually, the proof of knowledge may also yield a witness that C won one of the RK-iterations in the right session. However, because no rewinding of the right party is performed during the simulation, this can only happen with negligible probability.

[40] Formally, the simulator outputs a simulation of C_1's view, but C_1's output can be computed from its view.

$\frac{\text{poly}(n)}{p}$ number of times). The simulator will then use $T = \frac{4}{p}q(n)$ as a timeout value. That is, if the simulation takes more than T steps (which will happen with probability less than $\frac{1}{2}$) then the simulator, will restart from the beginning, where it will use at most n restarts. This will ensure expected polynomial-time simulation, and only introduce a negligible statistical bias.

□

6.5.2 The case of Zero-Knowledge

The analog for zero knowledge of Theorem 6.5.1 is the following:

Theorem 6.5.5. *Suppose that there exists a constant-round zero-knowledge proof for* **NP** *that is extractable with respect to synchronizing adversaries. Then, there exists a constant-round zero-knowledge proof for* **NP** *that is extractable with respect to adversaries that use arbitrary scheduling.*

The proof of Theorem 6.5.5 follows the proof of Theorem 6.5.1, and in fact is somewhat easier because we don't need to obtain a *strongly* extractable scheme in the case of zero-knowledge. As in the case of commitment schemes, we assume that there exists a c round zero-knowledge scheme Π_1 that is extractable w.r.t. synchronizing adversaries and we construct a $c + O(1)$-round zero knowledge proof system Π_2 that is extractable with respect to non-synchronizing adversaries. The scheme Π_2 will just be a variant of the [RK99] proof system, where when proving that some x is in some language M, the left and right party perform c RK-iterations and then the left party proves to the right party in WI that either $x \in M$ or that it won one of the RK-iterations. We'll combine the two schemes using a scheme Π. In the scheme Π the first message from the left party to the right party will be a string $y = \mathsf{Com}(b)$ for some $b \in \{1, 2\}$. The left and right party will then run the schemes Π_1 and Π_2 in parallel, where the left will prove to the right in the scheme Π_b that either $x \in M$ or $y = \mathsf{Com}(b)$. We omit the full details and proof , since they are nearly identical to the case of commitment schemes. We note that one can also use the constructions of [DDN91] to reduce the problem of constructing a zero-knowledge scheme to the problem of constructing a commitment scheme and vice versa.

6.6 Conclusions and Open Questions.

We have shown a coin-tossing protocol that allows to transform many protocols secure in the shared random string model to protocols that are secure in the plain man-in-the-middle model. Using this coin-tossing protocol, we gave a constant-round non-malleable non-malleable commitment scheme and a constant-round non-malleable zero-knowledge argument system. It seems that our coin-tossing protocol can be applied in other settings in the man-in-the-middle model. In particular, it may be that it can be used to give a protocol for general 2-party secure computation in the man-in-the middle setting.

Complexity assumptions. Throughout this chapter, we have used a subexponential hardness assumption (namely, that there exist some cryptographic primitives secure against 2^{n^ϵ}-sized circuits, where n is the security parameter). However note that we have *not* used the "complexity leveraging" technique [CGGM00] in this work, and hence in some sense this assumption is not inherent. The place where we used our assumption is in the construction of evasive sets, which we needed to be more efficient than the hardness assumption. However, in this construction the assumption was used for the purpose of derandomizing probabilistic machines. Thus it was sufficient to use any hardness bound $s(\cdot)$ such that under the assumption that $s(n)$-strong one-way functions exist, there is a pseudorandom generator mapping $o(\log s(n))$ bits to n bits (and hence one can derandomize polynomial-time algorithms in time $s(n)^{o(1)}$). Although any subexponential function satisfies this condition, one can use slower growing functions such as "half exponential" functions.[41] It is also known that for the purpose of derandomization it is sufficient to assume *worst-case* hardness (e.g., [NW88, IW97]). Thus, one can obtain the same result by assuming an exponential worst-case assumption (that implies that $\mathbf{BPP} = \mathbf{P}$) along with a "nice" super-polynomial hardness bound on the cryptographic primitives (e.g., $n^{\log n}$). An open question is to construct constant-round non-malleable protocols under the standard assumption of hardness against polynomial-sized circuits.

Strict polynomial time. The simulators presented in this work run in *expected* probabilistic polynomial-time. We believe that one can obtain simulators that run in *strict* probabilistic polynomial-time by using the commit-with-extract and proof of knowledge of Chapter 5. However, this will result in (an even more) complicated non-black-box simulator, and thus we have chosen not to pursue this path.

Generalization to more parties. In this work we limited ourselves to a setting with two honest parties and one adversary. A natural question is whether these results generalize to a more general setting in which there are more honest parties communicating through an adversarial channel. The most general setting is when the number of honest parties is an arbitrary polynomial in the security parameter. A less general setting is when the number of honest parties is some fixed polynomial in the security parameter.

Another direction is to try to apply our techniques (diagonalization combined with universal arguments/CS proofs, non-black-box proofs of security) to other problems in cryptography.

[41] A function $f(\cdot)$ is half-exponential if $f(f(n)) > 2^n$.

Appendix A

Construction of Universal Arguments

Summary: We present a construction of a universal argument scheme based on the existence of collision-resistent hash functions. We also show how to convert this scheme to a zero-knowledge or an Arthur-Merlin witness-indistinguishable scheme.

A.1 Overview

In this appendix, we prove the following theorem

Theorem A.1.1 (Theorem 2.4.4, restated). *Suppose that collision-resistent hash functions exist. Then for every $\epsilon > 0$, there exists a constant-round universal argument system in which the total size of the messages exchanged when proving a statement of length n is at most n^ϵ. Furthermore,*

1. *There exist such systems where the prover is zero-knowledge.*
2. *There exist such systems that are Arthur-Merlin (public-coins) and the prover is witness indistinguishable (WI).*

We start by proving the following theorem:

Theorem A.1.2. *Suppose that collision-resistent hash functions exist. Then for every $\epsilon > 0$, there exists a constant-round Arthur-Merlin universal argument system in which the total size of the messages exchanged when proving a statement of length n verifiable in time t is at most $n^\epsilon \mathrm{polylog}(t)$.*

We will later show how one can convert the system obtained from Theorem A.1.2 into a system satisfying either one of the properties of Theorem A.1.1.

A.2 Proof of Theorem A.1.2

Our construction of universal arguments adapts Kilian's construction [Kil92] in a straightforward manner. Our contribution is in the analysis of this construction. Unlike in the previous analysis (as in [Kil92] and [Mic94]), in establishing computational-soundness via contradiction, we cannot afford to derive a collision-forming circuit of size that is (at least) polynomial in the complexity

This appendix is based on the paper [BG01], which is joint work with Oded Goldreich.

of the designated prover (which may be exponential in the input length).[1] We need to derive collision-forming circuit of size that is polynomial in the input length. Indeed, doing so allows us to use standard collision-free hashing (rather than strong ones).

The analysis is further complicated by our desire to establish the "weak proof of knowledge" property, which is needed in for our main application.

Motivation. In order to explain the difficulty and its resolution, let us recall the basic construction of Kilian [Kil92] (used also by Micali [Mic94]), as adapted to our setting.

Our starting point is a $\mathbf{PCP}[\text{poly}, \text{poly}]$ system for $L_{\mathcal{U}} \in \mathbf{NEXP}$, which is used in the universal-argument system as follows. The verifier starts by sending the prover a hashing function. The prover constructs a PCP-proof/oracle (corresponding to the common input and its own auxiliary input), places the bits of this oracle at the leaves of a polynomial-depth full binary tree, and places in each internal node the hash-value obtained by applying the hashing function to the labels of its children. The prover sends the label of the root to the verifier, which responses by sending a random tape of the type used by the PCP-verifier. Both parties determine the queries corresponding to this tape, and the prover responds with the values of the corresponding leaves along with the labels of the vertices along the paths from these leaves to the root (as well as the labels of the siblings of these vertices). The verifier checks that this sequence of labels matches the corresponding applications of the hashing function, and also emulates the PCP-verifier. Ignoring (for now) the issue of prover's complexity, the problem is in establishing computational-soundness.

The naive approach is to consider what the prover does on each of the possible random-tapes sent to it. In case it answers consistently (i.e., with leaves labels that depend only on the leave location), we obtain a PCP-oracle and soundness follows by the soundness of the PCP scheme. On the other hand, inconsistent labels for the same leaf yield a (hashing) collision somewhere along the path to the root. However, in order to find such a collision, we must spend time proportional to the size of the tree, which yields contradiction only in case the hashing function is supposed to withstand adversaries using that much time. In case the tree is exponential (or even merely super-polynomial) in the security parameter, we derive contradiction only when using hashing functions of subexponential (respectively, super-polynomial) security.

The approach taken here is to consider each leaf separately rather than all leaves together. That is, the naive analysis distinguishes the case that the prover answers inconsistently on *some* leaf from the case it answer consistently on *all* leaves. Instead, we consider each leave separately, and distinguishes the case that the prover answers inconsistently on *this leaf* from the case it answer consistently on *this leaf*. Loosely speaking, we call a leaf good if the prover answers consistently on it, and observe that if a big fraction of the leaves are good then soundness follows by the soundness of the PCP scheme. In case sufficiently many leaves are not good, we obtain a collision by picking a random leave (hoping that it is not good) and obtaining inconsistent labels for it. This requires being able to uniformly select a random-tape that makes the PCP-verifier make the corresponding query, a property which is fortunately enjoyed by the relevant PCP systems.

We warn that the above is merely a rough description of the main idea in our analysis. Furthermore, in order to establish the proof-of-knowledge property of our construction, we need to rely on an analogous property of the PCP system (which again happens to be satisfied by the relevant PCP systems).

[1] Specifically, Kilian's construction [Kil92] uses a PCP system, and the contradiction hypothesis is shown to yield a collision-forming circuit that is always bigger than the size of the relevant PCP proof/witness. Instead, we show how to obtain a collision-forming circuit that is smaller than the relevant PCP-proof.

A.2.1 The PCP system in use

We first recall the basic definition of a PCP system. Loosely speaking, a probabilistically checkable proof (PCP) system consists of a probabilistic polynomial-time verifier having access to an oracle which represents a proof in redundant form. Typically, the verifier accesses only few of the oracle bits, and these bit positions are determined by the outcome of the verifier's coin tosses. It is required that if the assertion holds then the verifier always accepts (i.e., when given access to an adequate oracle); whereas, if the assertion is false then the verifier must reject with high probability (as specified in an adequate bound), no matter which oracle is used. The basic definition of the PCP setting is given in Item (1) below. Typically, the complexity measures introduced in Item (2) are of key importance, but not so in this work.

Definition A.2.1 (PCP – basic definition). 1. A *probabilistic checkable proof system (PCP) with error bound* $\epsilon : \mathbb{N} \to [0,1]$ *for a language* L is a probabilistic polynomial-time oracle machine (called *verifier*), denoted V, satisfying

Completeness: For every $x \in L$ there exists an oracle π_x such that V, on input x and access to oracle π_x, always accepts x.

Soundness: For every $x \notin L$ and every oracle π, machine V, on input x and access to oracle π, rejects x with probability at least $1 - \epsilon(|x|)$.

2. Let r and q be integer functions. The complexity class $\mathbf{PCP}_\epsilon[r(\cdot), q(\cdot)]$ consists of languages having a PCP system with error bound ϵ in which the verifier, on any input of length n, makes at most $r(n)$ coin tosses and at most $q(n)$ oracle queries.

Note that if L has a PCP system with error bound ϵ then $L \in \mathbf{PCP}_\epsilon[p(\cdot), p(\cdot)]$, for some polynomial p. Here we will only care that $L_{\mathcal{U}} \in \mathbf{NE}$ has a PCP system with and exponentially decreasing error bound (i.e., $\epsilon(n) = 2^{-n}$). Instead of caring about the refine complexity measures (of Item 2), we will care about the following *additional properties* satisfied by this specific PCP system, where only some of these properties were explicitly considered before (see discussion below).

Definition A.2.2 (PCP – auxiliary properties:). Let V be a PCP verifier with error $\epsilon : \mathbb{N} \to [0,1]$ for a language $L \in \mathbf{NEXP}$, and let R be a corresponding witness relation. That is, if $L \in \text{Ntime}(t(\cdot))$, then we refer to a polynomial-time decidable relation R satisfying $x \in L$ if and only if there exists w of length at most $t(|x|)$ such that $(x, w) \in R$. We consider the following auxiliary properties:

Relatively-efficient oracle-construction: This property holds if there exists a polynomial-time algorithm P such that, given any $(x, w) \in R$, algorithm P outputs an oracle π_x that makes V always accept (i.e., as in the completeness condition).

Non-adaptive verifier: This property holds if the verifier's queries are determined based only on the input and its internal coin tosses, independently of the answers given to previous queries. That is, V can be decomposed into a pair of algorithms, Q and D, such that on input x and random-tape r, the verifier makes the query sequence $Q(x,r,1), Q(x,r,2), ..., Q(x,r,p(|x|))$, obtains the answers $b_1, ..., b_{p(|x|)}$, and decides by according to $D(x, r, b_1 \cdots b_{p(|x|)})$.

Efficient reverse-sampling: This property holds if there exists a probabilistic polynomial-time algorithm S such that, given any string x and integers i and j, algorithm S outputs a uniformly distributed r that satisfies $Q(x, r, i) = j$, where Q is as above.

A proof-of-knowledge property: This property holds if there exists a probabilistic polynomial-time oracle machine E such that the following holds:[2] for every x and π, if $\Pr[V^\pi(x) = 1] > \epsilon(|x|)$ then there exists $w = w_1 \cdots w_t$ such that $(x, w) \in R$ and $\Pr[E^\pi(x, i) = w_i] > 2/3$ holds for every i.

Non-adaptive PCP verifiers were explicitly considered in several works, all popular PCP systems use non-adaptive verifiers, and in fact in some sources PCP is defined in terms of non-adaptive verifiers. The oracle-construction and proof-of-knowledge properties are implicit in some works, and are known to hold for most popular PCP systems (although to the best of our knowledge a proof of this fact has never appeared). To the best of our knowledge, the reverse-sampling property was not considered before. Nevertheless it can be verified that any $L \in \mathbf{NEXP}$ has a PCP system satisfying all the above properties.

Theorem A.2.3. *For every $L \in \mathbf{NEXP}$ and for every $\epsilon : \mathbb{N} \to [0, 1]$ such that $\epsilon(n) > 2^{-\mathrm{poly}(n)}$, there exist a PCP system with error ϵ for L that satisfies all properties in Definition A.2.2.*

Proof sketch: For $L \in \mathrm{Ntime}(t(\cdot))$, we consider a $\mathbf{PCP}_{1/2}[O(\log t(\cdot)), \mathrm{poly}(\cdot)]$ system as in [BFLS91](i.e., the starting point of [AS92, ALM+98]). (We stress that this PCP system, unlike the one of [FGL+91], uses oracles of length polynomial in t.)[3] This PCP system is non-adaptive and is well-known to satisfies the oracle-construction property. It is also known (alas less well-known) that this PCP system satisfies the proof-of-knowledge property. Finally, it is easy to see that this PCP system (as any reasonable PCP system we know of) also satisfies the reverse-sampling property.[4] Error reduction is obtained without effecting the oracle, and so it is easy to see that the amplified PCP preserves all the auxiliary properties. □

A.2.2 The actual construction

The construction is an adaptation of Kilian's construction [Kil92] (used also by Micali [Mic94]). Using Theorem A.2.3, we start with a PCP system with error $\epsilon(n) = 2^{-n}$ for $L_{\mathcal{U}}$ that satisfies the auxiliary properties in Definition A.2.2. Actually, the corresponding witness relation will not be $R_{\mathcal{U}}$ as defined in Section 2.4, but rather a minor modification of it, denoted $R'_{\mathcal{U}}$: the pair $((M, x, t), (w, 1^{t'}))$ is in $R'_{\mathcal{U}}$ if M accepts (x, w) in $t' \le t$ steps. (The purpose of this padding of the witness is to obtain a relation that is decidable in polynomial-time, as required in Definition A.2.2.) Let V_{PCP} denote the above PCP system (or its verifier), and $P_{\mathsf{PCP}}, Q_{\mathsf{PCP}}, D_{\mathsf{PCP}}, S_{\mathsf{PCP}}, E_{\mathsf{PCP}}$ denote the auxiliary algorithms (or machines) guaranteed by Definition A.2.2 (e.g., P_{PCP} is the oracle-constructing procedure, Q_{PCP} determines the verifier's queries, and S_{PCP} provides reverse-sampling).

A second ingredient used in the construction is a *random-access hashing* scheme (see Section 2.2.1). The basic idea is to have the prover use a random access hashing scheme to commit to the PCP proof. The construction of a universal argument scheme is depicted Protocol A.2.4.

Protocol A.2.4 is a constant-round Arthur-Merlin protocol. Clearly, Protocol A.2.4 satisfies the first two requirements of Definition 2.4.1; that is, the verifier's strategy is implementable in probabilistic polynomial-time, and completeness holds with respect to a prover strategy that (given $y = (M, x, t)$ and w as above) runs in time polynomial in $T_M(x, w)$. Furthermore, the communication complexity of this protocol only depends on the security parameter of the hashing function

[2] For negligible ϵ (as used below) this proof-of-knowledge property is stronger than the standard proof-of-knowledge property (as in [BG93] and [Gol01b, Sec. 4.7]).

[3] Moving to a non-binary encoding of objects seems important for achieving this.

[4] This property follows from the structure of the standard PCP systems. In our case, the system consists of a *sum-check* (a la Lund *et al.* [LFKN90]), and a *low-degree test*. In both tests, the queries are selected in a very simple manner, and what is complex (at least in the case of low-degree tests) is the analysis of the test.

Public input: $\langle M, x, t\rangle$ (statement to be proved is "$\exists_w$ s.t. $M(x;w)$ outputs 1 within t steps") **Prover's auxiliary input:** w (a witness that $M(x) = 1$)	$w \downarrow$ $\boxed{P}$ $\quad$ $M, x, t \downarrow$ $\quad$ $\boxed{V}$
Preliminary action by the prover. The prover invokes M on input (x, w), and obtains $t' = t_M(x, w)$. Assuming that $(y, w) \in R_{\mathcal{U}}$ and letting $w' = (w, 1^{t'})$, the prover obtains an $R'_{\mathcal{U}}$-witness; that is, $(y, w') \in R'_{\mathcal{U}}$. Invoking P_{PCP} on (y, w'), the prover obtains $\pi = P_{\mathsf{PCP}}(y, w')$.	
Step V1 (Choose hash-function): Verifier chooses a random seed α for a random access hash function collection $\{h_\alpha, \mathsf{cert}_\alpha\}_{\alpha \in \{0,1\}^*}$, and sends α to the prover.	$\xleftarrow{\alpha \leftarrow_{\mathrm{R}} \{0,1\}^n}$
Step P2 (Commit to PCP proof): Prover computes $z = h_\alpha(\pi)$ and sends it to the verifier.	$\xrightarrow{z = h_\alpha(\pi)}$
Step V3 (Choose PCP random tape): The verifier uniformly selects a random-tape r for the PCP system, and sends r to the prover.	$\xleftarrow{r \leftarrow_{\mathrm{R}} \{0,1\}^{r(n)}}$
Step P4 (Send PCP answers): The prover provides the corresponding (PCP) answers, augmented by *proofs of consistency* of these answers with the hash value provided in Step P2: First, invoking Q_{PCP}, the prover determines the sequence of queries that the PCP system makes on random-tape r. That is, for $i = 1, ..., m$, it computes $q_i = Q_{\mathsf{PCP}}(y, r, i)$, where $m \stackrel{def}{=} \mathrm{poly}(n)$ is the number of queries made by the system. Then, for $i = 1, ..., m$, the prover sends the q_i^{th} bit of π, along with $\mathsf{cert}_\alpha(\pi, i)$.	$\xrightarrow{\{\pi_{q_i}, \mathsf{cert}_\alpha(\pi, q_i)\}_{i=1}^m}$
Verifier decision The verifier checks that the answers provided by the prover would have been accepted by the PCP-verifier, and that the corresponding proofs of consistency are valid. That is, if we denote the prover's message by $b_1, \sigma_1, \ldots, b_m, \sigma_m$, then by invoking D_{PCP}, the verifier checks whether, on input y and random-tape r, the PCP-verifier would have accepted the answer sequence $b_1, \ldots, b_m$. That is, check whether $D_{\mathsf{PCP}}(y, r, b_1, \ldots, b_m) = 1$. Then, the verifier checks the consistency of the responses. That is, the verifier computes $q_i = Q_{\mathsf{PCP}}(y, r, i)$ for $i = 1, \ldots, m$, and runs $V_{\alpha,z}(q_i, b_i, \sigma_i)$, where V is the verification algorithm of the random access hashing scheme. The verifier accepts if and only if the prover's message passes all these checks.	

Protocol A.2.4. Universal Arguments

(along with a logarithmic dependence on the witness size), and thus by scaling this parameter appropriately, we can ensure that this complexity is at most $n^\epsilon \text{polylog}(t)$ for every fixed $\epsilon > 0$. We thus focus on establishing the two last requirements of Definition 2.4.1. In fact, computational soundness follows from the weak proof-of-knowledge property (because whenever some adversary can convince the verifier to accept with non-negligible probability the extractor outputs a valid witness for membership in $L_{\mathcal{U}}$). Thus, it suffices to establish the latter.

A.2.3 Establishing the weak proof-of-knowledge property

This subsection contains the main technical contribution of the current analysis over the previous analysis of Kilian and Micali [Kil92, Mic94]. The novel aspect in the analysis is the "local definition of a conflict" (i.e., a conflicting oracle-bit rather than a conflicting pair of oracles), and the fact that reverse-sampling can be used to derive (in polynomial-time) hashing-collisions (given a conflicting oracle bit-position).

Lemma A.2.5. *Protocol A.2.4 satisfies the weak proof-of-knowledge property of Definition 2.4.1, provided that the family $\{h_\alpha, \mathsf{cert}_\alpha\}_{\alpha\in\{0,1\}^*}$ is indeed a random access hashing scheme.*

Combining Lemma A.2.5 with the above discussion, we derive Theorem A.1.2.

Proof of Lemma A.2.5.

Fixing any polynomial p, we present a probabilistic polynomial-time knowledge-extractor that extracts witnesses from any feasible prover strategy that makes V accept with probability above the threshold specified by p. Specifically, for any family of (deterministic) polynomial-size circuits representing a possible cheating prover strategy and for all sufficiently long y's, if the prover convinces V to accept y with probability at least $1/p(|y|)$ then, with noticeable probability (i.e., $1/p'(|y|)$), the knowledge-extractor (given oracle access to the strategy) outputs the bits of a corresponding witness.

We fix an arbitrary family, $\{\widetilde{P}_n\}_{n\in\mathbb{N}}$, of (deterministic) polynomial-size circuits representing a possible cheating prover strategy, and a generic n and $y \in \{0,1\}^n$ such that $\Pr[(\widetilde{P}_n, V)(y) = 1] > \varepsilon \stackrel{def}{=} 1/p(n)$. We consider a few key notions regarding the interaction of $\widetilde{P}_n$ and the designated verifier V on common input y. First we consider notions that refer to a specific interaction (corresponding to a fixed sequence of verifier coins):

- The i^{th} query in such interaction is $q_i = Q_{\mathsf{PCP}}(y, r, i)$, where r is the value sent by the verifier in Step V3.
- The i^{th} answer supplied by (the prover) $\widetilde{P}_n$ is the but b_i it supplied its message of Step P4, that is, b_i is supposed to equal π_{q_i}. The corresponding authentication is the corresponding certificate σ_i.
- The i^{th} answer supplied by $\widetilde{P}_n$ is said to be proper if the corresponding authentication passes the verifier's test.

Next, we consider the probability distribution induced by the verifier's coins. Note that these coins consist of the pair of choices (α, r) that the verifier took in Steps V1 and V3, respectively. Fixing any $\alpha \in \{0,1\}^n$, we consider the conditional probability, denoted $p_{y,\alpha}$, that the verifier accepts when choosing α is Step V1. Clearly, for at least a $\varepsilon/2$ fraction of the possible α's it holds that $p_{y,\alpha} \geq \varepsilon/2$. We fix any such α for the rest of the discussion. We now consider notions that refer to the probability space induced by a uniformly chosen $r \in \{0,1\}^{\mathrm{poly}(n)}$ (selected by the verifier in Step V3).

- For any query $q \in \{0,1\}^d$, a query index $i \in \{1,...,m\}$, possible answer $b \in \{0,1\}$, and $\delta \in [0,1]$, we say that b is δ-strong for (i,q) if, conditioned on the i^{th} query being q, the probability that $\widetilde{P}_n$ *properly answers* the i^{th} with b is at least δ. That is,

$$\Pr_r[b_i = b \text{ is proper} \mid q_i = Q(y,r,i)] \ \geq \ \delta$$

  When i and q are understood from the context, we just say that b is a δ-strong answer.

- We say that a query $q \in \{0,1\}^d$ has δ-conflicting answers if there exist i and j (possibly $i = j$) such that 0 is δ-strong for (i,q) and 1 is δ-strong for (j,q).

We stress that throughout the rest of the analysis we consider a fixed $\alpha \in \{0,1\}^n$ and a uniformly distributed $r \in \{0,1\}^{\mathrm{poly}(n)}$.

Claim A.2.6. *The probability that the verifier accepts while receiving only δ-strong answers is at least $p_{y,\alpha} - m\delta$.*[5]

Thus, picking $\delta = p_{y,\alpha}/2m$, we may confine ourselves to the case that all the prover's answers are δ-strong.

Proof of Claim A.2.6. The key observation is that whenever the verifier accepts, all answers are proper. Intuitively, answers that are not δ-strong (i.e., are rarely proper) are unlikely to appear in such interactions. Specifically, we just upper bound the probability that, for some $i \in \{1,...,m\}$, the answer ℓ'_{q_i} is proper but not δ-strong for (i,q_i), where $q_i = Q(y,r,i)$. Fixing any i and any possible value of $q_i = Q(y,r,i)$, by definition (of being proper but not δ-strong), the corresponding event occurs with probability less than δ. Averaging over the possible values of $q_i = Q(y,r,i)$ we are done. $\square$

Claim A.2.7. *There exist a probabilistic polynomial-time oracle machine that, given α and oracle access to $\widetilde{P}_n$, finds two conflicting certificates to the random-access hashing scheme with probability that is polynomially related to δ/n and to the probability that the verifier makes a query that has δ-conflicting answers. Specifically, letting η denote the latter probability, the probability of finding a collision is at least $\eta\delta^2/m^3$.*

Thus, on a typical α, the probability η must be negligible (because otherwise we derive a contradiction to the security of the family $\{h_\alpha, \mathsf{cert}_\alpha\}_{\alpha\in\{0,1\}^*}$). Consequently, for $\delta = p_{y,\alpha}/2m > 1/\mathrm{poly}(n)$, we may confine ourselves to the case that the prover's answers are not $(\delta/2)$-conflicting.

Proof of Claim A.2.7. We uniformly select $r \in \{0,1\}^{\mathrm{poly}(n)}$ and $i \in \{1,...,m\}$, hoping that $q_i = Q(y,r,i)$ is δ-conflicting (which is the case with probability at least η/m). Uniformly selecting $i', i'' \in \{1,...,m\}$, and invoking the reverse-sampling algorithm S_{PCP} on inputs (y,i',q_i) and (y,i'',q_i), respectively, we obtain uniformly distributed r' and r'' that satisfy $q_i = Q(y,r',i')$ and $q_i = Q(y,r'',i'')$. We now invoke $\widetilde{P}_n$ twice, feeding it with α and r' (resp. α and r'') in the first (resp., second) invocation. With probability at least $(\delta/m)^2$ both answers to q_i will be proper but with opposite values. $\square$

[5]Recall that m is the number of queries asked by the PCP verifier V_{PCP}.

Suppose for a moment, that (for $\delta = p_{y,\alpha}/2m$) all the prover's answers are δ-strong but not $(\delta/2)$-conflicting. Then, we can use the prover's answers in order to construct (and not merely claim the existence of) an oracle for the PCP system that makes it accept with probability at least $p_{y,\alpha}/2$. Specifically, let the q^{th} bit of the oracle be b if and only if there exists an i such that b is δ-strong for (i, q). The setting of the oracle bits can be decided in probabilistic polynomial-time by using the reverse-sampling algorithm S_{PCP} to generate multiple samples of interactions in which these specific oracle bits are queried. That is, to determine the q^{th} bit, we try $i = 1, ..., m$, and for each value of i generate multiple samples of interactions in which the i^{th} oracle query equals q. We will use the gap provided by the hypothesis that for some i there is an answer that is δ-strong for (i, q), whereas (by the non-conflicting hypothesis) for every j the opposite answer is not $(\delta/2)$-strong for (j, q).

In general, in contrary to the simplifying assumption above, some queries may either have no strong answers or be conflicting. The procedure may indeed fail to recover the corresponding entries in the PCP-oracle, but this will not matter because with sufficient high probability the PCP verifier will not query these badly-recovered locations.

The oracle-recovery procedure: We present a probabilistic polynomial-time oracle machine that on input (y, α) and $q \in \{0, 1\}^d$ and oracle access to the prover $\widetilde{P}_n$, outputs a candidate for the q^{th} bit of a PCP-oracle. The procedure operates as follows, where $T \stackrel{def}{=} \text{poly}(n/\delta)$ and $\delta = \varepsilon/4m$:

1. For $i = 1, ..., m$ and $j = 1, ..., T$, invoke S_{PCP} on input (y, i, q) and obtain $r_{i,j}$.
2. For $i = 1, ..., m$ and $j = 1, ..., T$, invoke $\widetilde{P}_n$ feeding it with α and $r_{i,j}$, and if the i^{th} answer is proper then *record* (i, j) *as supporting this answer value.*
3. If for some $i \in \{1, ..., m\}$, there are $(2\delta/3) \cdot T$ records for the form $(i, \cdot)$ for value $b \in \{0, 1\}$ then define b as a *candidate*. That is, b is a candidate if there exists an i and at least $(2\delta/3) \cdot T$ different j's such that the i^{th} answer of $\widetilde{P}_n(\alpha, r_{i,j})$ is proper and has value b.
4. If a single value of $b \in \{0, 1\}$ is defined as a candidate then set the q^{th} bit accordingly. (Otherwise, do whatever you please.)

We call the query q good if it does not have $(\delta/2)$-conflicting answers and there exists an $i \in \{1, ..., m\}$ and a bit value that is δ-strong for (i, q). For a good query, with overwhelmingly high probability, the above procedure will define the latter value as a unique candidate. (The expected number of $(i, \cdot)$-supports for the strong value is at least $\delta \cdot T$, whereas for the opposite value the expected number of $(i', \cdot)$-supports is less than $(\delta/2) \cdot T$, for every i'.) Let use denote the PCP-oracle induced by the above procedure by π.

Claim A.2.8. *Let $\delta = \varepsilon/4m$ and recall that $p_{y,\alpha} \geq \varepsilon/2$. Suppose that the probability that V makes a query that has $(\delta/2)$-conflicting answers is at most $p_{y,\alpha}/4$. Then, with probability at least $1 - 2^{-n}$ taken over the reconstruction of π, the probability that ${V_{\mathsf{PCP}}}^{\pi}(y)$ accepts is lower bounded by $p_{y,\alpha}/4$.*

Proof of Claim A.2.8. Combining the hypothesis (regarding $(\delta/2)$-conflicting answers) with Claim A.2.6, we conclude that with probability at least $(p_{y,\alpha} - m\delta) - (p_{y,\alpha}/4) \geq p_{y,\alpha}/4$ the verifier (of the interactive argument) accepts while making only good queries and receiving only δ-strong answers. However, in this case, with probability at least $1 - 2^{-n}$, the answers of $\widetilde{P}_n$ equal the corresponding bits in π. (This is because for a good query, with overwhelmingly high probability, the answer that is δ-strong will be the only candidate determined by the procedure.) Since the (interactive argument) verifier is accepting after invoking V_{PCP} on the answers it obtained, it follows that in this case V_{PCP} accepts too. □

The weak proof-of-knowledge property (of the interactive argument) now follows from the corresponding property of the PCP system. Specifically, we combine the PCP extractor with the above oracle-recovery procedure, and obtain the desired extractor.

Extractor for the argument system: On input (y, i) and access to a prover strategy $\widetilde{P}_n$, the extractor operates as follows (using $\delta = \varepsilon/4m$):

1. Uniformly select $\alpha \in \{0,1\}^n$, hoping that α is typical (with respect to collision-freeness) and that $p_{y,\alpha} > \varepsilon/2$ (which hold with probability at least $(\varepsilon/2) - \mu(n) > \varepsilon/3$).

   By saying that α is typical with respect to collision-freeness we mean that in the corresponding conditional probability space (of α being chosen in Step V1), the probability that the verifier makes a query that has $(\delta/2)$-conflicting answers is less than $p_{y,\alpha}/4$, which in turn is at least $\varepsilon/8$. By Claim A.2.7, there exists a probabilistic polynomial-time machine that, given any untypical α (and access to $\widetilde{P}_n$), finds conflicting certificates for $h_\alpha, \mathsf{cert}_\alpha$ with probability at least $(p_{y,\alpha}/4) \cdot (\delta/2)^2/m^3 = \Omega(\varepsilon^3/m^5) = 1/\mathrm{poly}(n)$, because $\varepsilon = 1/p(n)$. It follows that the fraction of untypical α's must be negligible.

2. Uniformly select coins ω for the oracle-recovery procedure, and fix ω for the rest of the discussion.

   Note that the oracle-recovery procedure (implicitly) provides oracle access to a PCP-oracle π, where (by Claim A.2.8) with probability at least $1 - 2^{-n}$ (over the choice of ω), this π convinces V_{PCP} with probability at least $p_{y,\alpha}/4 > \varepsilon/8 > 2^{-n}$.

3. Invoke $E_{\mathsf{PCP}}(y, i)$ providing it with oracle access to π. This means that each time $E_{\mathsf{PCP}}(y, i)$ makes a query q, we invoke the oracle-recovery procedure on input (y, q) (and with α and ω as fixed above), and obtain the q^{th} bit of π, which we return as answer to $E_{\mathsf{PCP}}(y, i)$. When $E_{\mathsf{PCP}}(y, i)$ provides an answer (supposedly the i^{th} bit of a suitable witness w), we just output this answer.

Let us call α useful if it is typical w.r.t collisions and $p_{y,\alpha} > \varepsilon/2$. We say that ω is α-useful if when used as coins for the oracle-recovery procedure (which gets main-input (y, α)) yields an oracle that convinces V_{PCP} with probability at least 2^{-n}. Recall that at least a $\varepsilon/3$ fraction of the α's are useful, and that for each useful α at least a $1 - 2^{-n}$ fraction of the ω's are α-useful. By the proof-of-knowledge property of the PCP system, if α is useful and ω is α-useful then with probability at least $2/3$ (over the coins of E_{PCP}), the output of E_{PCP} (and thus of our extractor) will be correct. By suitable amplification, we can obtain the correct answer with probability at least $1 - 2^{-2n}$ (over the coins of E_{PCP}), pending again on α and ω being useful. Denoting the coins of the amplified E_{PCP} by ρ, we conclude that for at least a fraction $1 - t \cdot 2^{-2n} \geq 1 - 2^{-n}$ of the possible ρ's, the amplified E_{PCP} provides correct answers for all t bit locations. We call such ρ's (α, ω)-useful.

Let us denote the above extractor by E. The running-time of E is dominated by the running-time of the oracle-recovery procedure, whereas the latter is dominated by the $\mathrm{poly}(n/\varepsilon)$ invocations of $\widetilde{P}_n$ (during the oracle-recovery procedure). Using $\varepsilon = 1/p(n)$, it follows that E runs in polynomial-time. The random choices of E correspond to the above three steps; that is, they consist

Public input: $\langle M, x, t\rangle$ (statement to be proved is "$\exists_w$ s.t. $M(x;w)$ outputs 1 within t steps") **Prover's auxiliary input:** w (a witness that $M(x) = 1$)	$w \downarrow$ $M, x, t \downarrow$ $\boxed{P}$ $\boxed{V}$
Steps P,V1.x ("Encrypted" UA proof): The prover and the verifier engage in the universal argument for $\langle M, x, t\rangle$ using the system of Protocol A.2.4. However, the prover does not send its messages in the clear, but rather only sends a *commitment* to these messages, using a simple commitment Com. The verifier sends its messages in the clear (note that since Protocol A.2.4 an Arthur-Merlin protocol, the verifier can compute its messages without knowing the prover's messages). We denote the transcript of this execution by τ, and the coins used by the prover in constructing the commitment by d.	$w \downarrow$ $\langle M, x, t\rangle \downarrow$ "Encrypted" UA $\langle M, x, t\rangle \in L_{\mathcal{U}}$ $\downarrow \tau$
Steps P,V2.x (Zero-knowledge proof of knowledge): The prover proves to the verifier using a zero-knowledge proof (or argument) of knowledge for **NP** that the transcript τ contains commitments to an accepting transcript for the universal argument system. Note that, unlike "$\langle M, x, t\rangle \in L_{\mathcal{U}}$", this is in **NP** statement and so can be proven using the zero-knowledge system.	$d \downarrow$ $\tau \downarrow$ $ZKPOK$ $\mathsf{Com}^{-1}(\tau)$ is accepting. $\downarrow$ 0/1

Protocol A.3.1. Zero-Knowledge Universal Arguments

of α, ω and ρ. Whenever they are all useful (i.e., α is useful, ω is α-useful, and ρ is (α, ω)-useful), the extractor E recovers correctly all bits of a suitable witness (for y). The event in the condition occurs with probability at least $(\varepsilon/3) \cdot (1 - 2^{-n}) \cdot (1 - 2^{-n}) > \varepsilon/4 = 1/4p(n)$. Letting $p'(n) = 4p(n)$, Lemma A.2.5 follows. $\square$

A.3 Making the Universal Arguments Zero Knowledge

We now use Theorem A.1.2 to prove Item 1 of Theorem A.1.1. That is, we show how to convert a universal argument systems into a proof system, where the prover is zero-knowledge, using any zero-knowledge proof of knowledge for **NP**. The transformation follows the ideas of a similar transformation by Kilian [Kil92].

Clearly Protocol A.3.1 satisfies the verifier efficiency and completeness with a relatively efficient prover requirements of Definition 2.4.1. Also, because the communication complexity of the second stage has a fixed polynomial dependence on the communication complexity of the first stage, we can ensure that the total communication complexity is at most $n^{\epsilon}\mathrm{polylog}(t)$. Note also that the total number of communication rounds of Protocol A.3.1 is constant.

We will now sketch why Protocol A.3.1 satisfies the zero-knowledge and weak proof of knowledge properties.

Zero Knowledge The simulator for the system works by sending commitments to "junk" messages (e.g., the all-zeros string) in the first stage, and then using the simulator for the **NP** zero-knowledge system in the second stage. One can prove that the output of this simulator is indeed indistinguishable from the verifier's view in a real interaction using the hybrid argument. That is, we consider a "hybrid simulator" which uses the honest universal argument prover in the first stage (instead of committing to "junk" messages) but uses the simulator (and not the honest prover) in the second stage. The security of the commitment scheme, along with the hiding property of the simple commitment scheme, ensure that both the actual simulator's output, and the real view of the verifier are indistinguishable from the output of the hybrid simulator.

Weak proof of knowledge To prove the weak proof of knowledge property, we show the following: for every (possibly cheating) prover P^* that causes the honest verifier of Protocol A.3.1 to accept with probability ϵ, there exists a prover P^{**} that causes the honest verifier of the original universal argument system (i.e., Protocol A.2.4) to accept with probability ϵ' which is polynomially related to ϵ. Transforming the prover P^* into the prover P^{**} can be done in probabilistic polynomial-time. The idea behind this transformation is that P^{**} will execute an internal copy of P^*, and will attempt to simulate the execution of P^* in the first stage, except that P^{**} will need to send the prover messages in the clear, whereas P^{**} sends only *commitments* to these messages. However, P^{**} can use the *knowledge extractor* of the second stage system to obtain the plaintext for these messages. Because the number of rounds in the universal arguments system is constant, it is actually not hard to show that P^{**} will indeed succeed with probability at least ϵ^c for some constant c.

To be a little more detailed, let P^* denote a cheating prover strategy that causes the honest verifier of Protocol A.3.1 to accept with probability ϵ. Using a Markov argument, we see that for at least an $\epsilon/2$ fraction of the choices for verifie's first message m_1 for the first round, the acceptance probability conditioned on the verifier's first message being m_1 is at least $\epsilon/2$. We call all such choices for the verifier's first message *good*. Similarly, conditioned on m_1 being a good verifier message, for an $\epsilon/4$ fraction of the choices for the verifier's second message m_2, the probability of acceptance conditioned on the first two verifier messages being $\langle m_1, m_2\rangle$ is at least $\epsilon/4$. We call these choices for the verifier's second message m_1-*good* or, when m_1 is clear from the context, simply *good*. We can continue in a similar way to define what it means for a choice of message of any round to be good. We say that a transcript $\langle m_1, \dots, m_l\rangle$ of all the verifier messages in the proof system is *good* if for every i, the message m_i is $\langle m_1, \dots, m_{i-1}\rangle$-good. Note that, because the number of rounds in Protocol A.3.1 is constant, there is at least an $\epsilon^{\Omega(1)}$ probability for a transcript to be good, if all verifier's messages are chosen according to the honest verifier strategy. We'll now assume for simplicity that the zero-knowledge proof system for **NP** has the property that given the prover's accepting answers for two verifier's message sequence of the form $\langle m_1, \dots, m_{i-1}, m_i\rangle$ and $\langle m_1, \dots, m_{i-1}, m'_i\rangle$ (with $m_i \neq m'_i$) one can obtain a witness for the **NP** statement (there are zero-knowledge systems for **NP** with this property). Now, given any prefix of *good* verifier messages, the verifier has $\epsilon^{\Omega(1)}$ probability to be able to complete this prefix to two such continuations. We can now see how the external prover P^{**} will behave: Whenever P^* receives such a message, it will feed this message to the internal prover P^* and obtain a *commitment* z to a prover answer in the original UA system. P^* will then try to compute two continuations of the form above to that message, and feed these two continuations to the internal prover P^*, to obtain the plaintext of the commitment z. It will then send this plaintext to the verifier. Assuming that all the verifier's messages and all the continuations are good (which will happen with $\epsilon^{\Omega(1)}$ probability, because the number of rounds is constant), the external prover P^{**} will convince the honest verifier of the original UA system to accept.

A.4 Making the Universal Arguments Witness Indistinguishable

Protocol A.3.1 is a zero-knowledge universal arguments, and so in particular it is a witness-indistinguishable universal argument. However, to prove Item 2 of Theorem A.1.1, we need to construct a (constant-round) *Arthur-Merlin* witness-indistinguishable proof system. Protocol A.3.1 would indeed satisfy this condition *if* the zero-knowledge proof for **NP** used in its second stage is constant-round Arthur-Merlin. Indeed, we show such a system exists in Chapter 4. However, we can not use this system in proving Theorem A.1.1, since this theorem is used in the construction of the system. Thus, we need a direct way to use a witness-indistinguishable proof of knowledge for **NP** to convert the universal argument system of Protocol A.2.4 to a witness-indistinguishable universal argument. This transformation is quite similar (but not identical) to the transformation we used in the previous section (Section A.3). It is presented in Protocol A.4.1. The main idea is to have the prover use an "encrypted" universal arguments *twice*, and then to prove using a WI system for **NP** that in *one* of these times the transcript was valid.

All properties of this system are the same as Protocol A.3.1, thus the only property we need to prove is the witness indistinguishability property.[6] We will only sketch why it holds. Let $u = \langle M, x, t\rangle$ be a statement and let w_1 and w_2 be two witnesses that $u \in L_{\mathcal{U}}$. We need to show that the verifier's view when the prover uses w_1 as auxiliary input is indistinguishable from its view when the prover uses w_2. We do so by considering several intermediate hybrids:

Hybrid $\mathbf{H_1}$: The first hybrid we consider, $\mathbf{H_1}$, is the verifier's view in a real interaction with the prover, when the prover uses w_1 as auxiliary input.

Hybrid $\mathbf{H_2}$: The second hybrid we consider, $\mathbf{H_1}$, is the verifier's view in an interaction where the prover uses the second witness w_2 in the second execution of the "encrypted" universal argument stage. This is indistinguishable from $\mathbf{H_1}$ by the hiding property of the commitment scheme.

Hybrid $\mathbf{H_3}$: In $\mathbf{H_3}$, we change the view from $\mathbf{H_2}$ by having the prover use the decommitment information for the *second* execution in the WI **NP** proof of the second stage. This is indistinguishable from $\mathbf{H_2}$ by the WI property of the WIPOK for **NP**.

Hybrid $\mathbf{H_4}$: In $\mathbf{H_4}$, we change the view from $\mathbf{H_3}$ by having the prover use w_1 for both the first and second executions of the "encrypted" universal argument stage. This is indistinguishable from $\mathbf{H_3}$ by the hiding property of the simple commitment scheme.

Hybrid $\mathbf{H_5}$: In $\mathbf{H_5}$, we change the view from $\mathbf{H_4}$ by having the prover use the decommitment information from the *first* execution in the WIPOK of the second stage. This is indistinguishable from $\mathbf{H_4}$ by the WI property of the WIPOK for **NP**. However, this is exactly the verifier's view in an interaction with the prover when the prover uses w_2 as auxiliary input.

[6]The only difference between this case and the proof sketched in Section A.3 gis that when proving the proof of knowledge, the external prover P^{**} will need to "guess" which of the two encrypted proofs is the valid one. However, it will guess correctly with probability $\frac{1}{2}$.

Public input: $\langle M, x, t\rangle$ (statement to be proved is "$\exists_w$ s.t. $M(x; w)$ outputs 1 within t steps") **Prover's auxiliary input:** w (a witness that $M(x) = 1$)	$w \quad M, x, t$ $\downarrow \quad \downarrow$ $\boxed{P} \quad \boxed{V}$
Steps P,V1.x ("Encrypted" UA proof): The prover and the verifier engage *twice* the universal argument for $\langle M, x, t\rangle$ using the system of Protocol A.2.4. However, the prover does not send its messages in the clear, but rather only sends a *commitment* to these messages, using a simple commitment Com. The verifier sends its messages in the clear. We denote the transcript of the first (resp. second) execution by τ_1 (resp. τ_2), and the coins used by the prover in constructing the commitments of the first (resp. second) execution by d_1 (resp. d_2).	$w \quad \langle M, x, t\rangle$ $\downarrow \quad \downarrow$ "Encrypted" UA $\langle M, x, t\rangle \in L_{\mathcal{U}}$ $\downarrow \quad \downarrow$ $d_1 \quad \tau_1$ $w \quad \langle M, x, t\rangle$ $\downarrow \quad \downarrow$ "Encrypted" UA $\langle M, x, t\rangle \in L_{\mathcal{U}}$ $\downarrow \quad \downarrow$ $d_2 \quad \tau_2$
Steps P,V2.x (Witness-indistinguishable proof.): The prover proves to the verifier using a witness-indistinguishable proof of knowledge for **NP** that either the transcript τ_1 contains commitments to an accepting transcript for the universal argument system, or the transcript τ_2 contains such commitments. The prover uses d_1 (the coins used in the first execution) to prove the statement.	$d_1, d_2 \quad \tau_1, \tau_2$ $\downarrow \quad \downarrow$ $WIPOK$ $\mathsf{Com}^{-1}(\tau_1)$ or $\mathsf{Com}^{-1}(\tau_2)$ is accepting. $\downarrow$ 0/1

Protocol A.4.1. Witness-Indistinguishable Universal Arguments

This completes the proof of Theorem A.1.1. We note that if one wants to conserve rounds, then it is possible to run the two executions of the "encrypted" universal argument stage in *parallel*.

□

Why use two encrypted proofs? One can see that we need to use two encrypted proofs to make the current hybrid argument work. We do not know whether this protocol will be witness indistinguishable even if we use a single encrypted proof. We *do* know that this protocol will be secure if the proof system for **NP** used in the second stage satisfies a stronger notion than witness indistinguishability that is called *strong* witness indistinguishability (sWI) [Gol01b, Sec 4.6]. Indeed, in the extended abstract presenting this transformation [BG01], we used a single encrypted proof and an sWI proof for **NP** to obtain a WI universal argument. However, it turns out that, unlike stated in [Gol01b, Sec 4.6], sWI is *not* closed under parallel composition, see the errata for Goldreich's book [Gol04, App C].[7] This implies that, unlike claimed in [BG01], the parallel versions of the well-known 3-round zero-knowledge arguments for 3-coloring and Hamiltonicity are *not* known to be strong-WI. We do not know of a constant-round Arthur-Merlin strong-WI proof or argument for **NP** other than the argument that is obtained from Chapter 4, and this is why we use the current construction, with two encrypted proofs.

[7]This errata is also available on `http://www.wisdom.weizmann.ac.il/~oded/foc-vol1.html\#err` .

Bibliography

[Ame00] Entry: black box. In J. P. e. a. Pickett, editor, *The American Heritage Dictionary of the English Language, 4th Edition*. Boston: Houghton Mifflin Company, 2000. `http://www.bartleby.com/61/`.

[Abr16] A. Abrams. *New Concepts in Diagnosis and Treatment*. 1916. See `http://www.meridianinstitute.com/eaem/abrams1/abr1cont.html`.

[ACGS84] W. Alexi, B. Z. Chor, O. Goldreich, and C.-P. Schnorr. RSA and Rabin Functions: Certain Parts are as Hard as the Whole. *SIAM J. Comput.*, 17(2):194–209, Apr. 1988. Preliminary version in FOCS' 84.

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof Verification and the Hardness of Approximation Problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs; a new characterization of NP. In *Proc.* 33*rd FOCS*, pages 2–13. IEEE, 1992.

[BFLS91] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. In *Proc.* 22*nd STOC*, pages 21–31. ACM, 1991.

[BM88] L. Babai and S. Moran. Arthur-Merlin Games: A Randomized Proof System and a Hierarchy of Complexity Classes. *J. Comput. Syst. Sci.*, 36:254–276, 1988.

[Bar01] B. Barak. How to go beyond the black-box simulation barrier. In *Proc.* 42*nd FOCS*, pages 106–115. IEEE, 2001. Preliminary full version available on `http://www.math.ias.edu/~boaz`.

[Bar02] B. Barak. Constant-Round Coin-Tossing With a Man in the Middle or Realizing the Shared Random String Model. In *Proc.* 43*rd FOCS*. IEEE, 2002. Preliminary full version available on `http://www.math.ias.edu/~boaz`.

[BG01] B. Barak and O. Goldreich. Universal Arguments and their Applications. Cryptology ePrint Archive, Report 2001/105, 2001. Extended abstract appeared in CCC' 2002.

[BGGL01] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell. Resettably-Sound Zero-Knowledge and its Applications. Record 2001/063, Cryptology ePrint Archive, Aug. 2001. Preliminary version appeared in FOCS' 01.

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahay, S. Vadhan, and K. Yang. On the (Im)possibility of Obfuscating Programs. In *Crypto '01*, 2001. LNCS No. 2139.

[BL02] B. Barak and Y. Lindell. Strict Polynomial-time in Simulation and Extraction. Cryptology ePrint Archive, Report 2002/043, 2002. Extended abstract appeared in STOC' 02.

[BOV03] B. Barak, S. J. Ong, and S. Vadhan. Derandomization in Cryptography, 2003.

[BG93] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. *Lecture Notes in Computer Science*, 740:390–420, 1993.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First Annual Conference on Computer and Communications Security*. ACM, November 1993.

[Blu82] M. Blum. Coin Flipping by Phone. In *Proc. 24th IEEE Computer Conference (CompCon)*, pages 133–137, 1982. See also *SIGACT News*, Vol. 15, No. 1, 1983.

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract). In *Proceedings of 20^{th} STOC*, pages 103–112, 2–4 May 1988.

[BG84] M. Blum and S. Goldwasser. An *Efficient* Probabilistic Public-Key Encryption Scheme which Hides All Partial Information. In *Crypto '84*, pages 289–299, 1984. LNCS No. 196.

[BL96] D. Boneh and R. Lipton. Algorithms for Black-Box Fields and their Applications to Cryptography. In *Crypto '96*, pages 283–297, 1996. LNCS No. 1109.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, Oct. 1988.

[BCY89] G. Brassard, C. Crépeau, and M. Yung. Everything in NP Can Be Argued in *Perfect* Zero-Knowledge in a *Bounded* Number of Rounds. In *Eurocrypt '89*, pages 192–195, 1989. LNCS No. 434.

[CF01] R. Canetti and M. Fischlin. Universally Composable Commitments. Report 2001/055, Cryptology ePrint Archive, July 2001. Extended abstract appeared in CRYPTO 2001.

[CGGM00] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable Zero-Knowledge. In *Proc. 32th STOC*, pages 235–244. ACM, 2000.

[CGH98] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In *Proc. 30th STOC*, pages 209–218. ACM, 1998.

[CKPR01] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires $\tilde{\Omega}(\log n)$ Rounds. Record 2001/051, Cryptology ePrint Archive, June 2001. Extended abstract appeared in STOC' 01.

[CR87] B. Chor and M. O. Rabin. Achieving independence in logarithmic number of rounds. In *Proc. 6th ACM PODC*, pages 260–268. ACM, 1987.

[Clo03] CloakWare Corporation. Introduction to CloakWare/Transcoder, Version 2.0, 2003. Available on `http://206.191.60.52/products/transcoder.html`, last visited on December 2003.

[CT00] C. Collberg and C. Thomborson. Watermarking, Tamper-Proofing, and Obfuscation – Tools for Software Protection. Technical Report TR00-03, The Department of Computer Science, University of Arizona, Feb. 2000.

[CTL97] C. Collberg, C. Thomborson, and D. Low. A Taxonomy of Obfuscating Transformations. Technical Report 148, University of Auckland, July 1997. See also `http://www.cs.arizona.edu/~collberg/Research/Obfuscation/index.html`.

[DDO+01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust Non-interactive Zero Knowledge. In *CRYPTO ' 2001*, pages 566–598, 2001.

[DIO98] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Non-interactive and non-malleable commitment. In *Proc.* 30*th STOC*, pages 141–150. ACM, 1998.

[DKOS01] G. Di Crescenzo, J. Katz, R. Ostrovsky, and A. Smith. Efficient and Non-Interactive Non-Malleable Commitment. Report 2001/032, Cryptology ePrint Archive, Apr. 2001. Preliminary versoin in EUROCRYPT 2001.

[DH76] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, Nov. 1976.

[DDN91] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437 (electronic), 2000. Preliminary version in STOC 1991.

[DNRS99] C. Dwork, M. Naor, O. Reingold, and L. Stockmeyer. Magic functions. In *Proc.* 40*th FOCS*, pages 523–534. IEEE, 1999.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero Knowledge. In *Proc.* 30*th STOC*, pages 409–418. ACM, 1998.

[Edw00] H. Edwards. Radionics, Good for Everything. In *The Skeptic Journal*, volume 13. Australian Skeptic Society, 2000. `http://www.skeptics.com.au/journal/radionics.htm`.

[FLS99] Feige, Lapidot, and Shamir. Multiple Noninteractive Zero Knowledge Proofs Under General Assumptions. *SIAM J. Comput.*, 29, 1999.

[Fei90] U. Feige. *Alternative Models for Zero Knowledge Interactive Proofs*. PhD thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 1990.

[FFS87] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *J. Cryptology*, 1(2):77–94, 1988. Preliminary version in STOC 1987.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive Proofs and the Hardness of Approximating Cliques. *J. ACM*, 43(2):268–292, Mar. 1996. Preliminary version in STOC 1991.

[FS89] U. Feige and A. Shamir. Zero Knowledge Proofs of Knowledge in Two Rounds. In *Crypto '89*, pages 526–545, 1989. LNCS No. 435.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proc.* 22*nd STOC*, pages 416–426. ACM, 1990.

[FM91] J. Feigenbaum and M. Merritt. *Distributed computing and cryptography: proceedings of a DIMACS Workshop, October 4–6, 1989*, volume 2 of *DIMACS series in discrete mathematics and theoretical computer science*. 1991.

[FS86] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Crypto '86*, pages 186–194, 1986. LNCS No. 263.

[FF00] M. Fischlin and R. Fischlin. Efficient Non-malleable Commitment Schemes. *Lecture Notes in Computer Science*, 1880:413–430, 2000. Extended abstract in CRYPTO' 2000.

[GT00] R. Gennaro and L. Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *Proc. 41st FOCS*, pages 305–313. IEEE, 2000.

[Gol97] Goldreich. Notes on Levin's Theory of Average-Case Complexity. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1997. ECCC Report TR97-058.

[Gol93] O. Goldreich. A Uniform-Complexity Treatment of Encryption and Zero-Knowledge. *J. Cryptology*, 6(1):21–53, 1993.

[Gol01a] O. Goldreich. Concurrent Zero-Knowledge With Timing, Revisited. Technical Report 2001/104, Cryptology ePrint Archive, Nov. 2001. Extended abstract in STOC' 02.

[Gol01b] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001. See also lecture notes available on `http://www.wisdom.weizmann.ac.il/~oded/frag.html` .

[Gol04] O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004. To appear in Spring 2004, draft available on `http://www.wisdom.weizmann.ac.il/~oded/foc-vol2.html#err` .

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. Preliminary version in FOCS' 84.

[GK96] O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *J. Cryptology*, 9(3):167–189, Summer 1996.

[GK92] O. Goldreich and H. Krawczyk. On Sparse Pseudorandom Ensembles. *Random Structures and Algorithms*, 3(2):163–174, 1992.

[GK90] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM J. Comput.*, 25(1):169–192, Feb. 1996. Preliminary version appeared in ICALP' 90.

[GL89] O. Goldreich and L. A. Levin. A Hard-Core Predicate for All One-Way Functions. In *Proc. 21st STOC*, pages 25–32. ACM, 1989.

[GL00] O. Goldreich and Y. Lindell. Session-Key Generation using Human Passwords Only. Report 2000/057, Cryptology ePrint Archive, Nov. 2000. Extended abstract in CRYPTO 2001.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *J. ACM*, 38(3):691–729, July 1991. Preliminary version in FOCS' 86.

[GO87] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *J. Cryptology*, 7(1):1–32, Winter 1994. Preliminary version in FOCS' 87.

[GO96] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

[GM82] S. Goldwasser and S. Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, Apr. 1984. Preliminary version appeared in STOC' 82.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. Preliminary version in STOC' 85.

[GMR84] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, 17(2):281–308, Apr. 1988. Preliminary version in FOCS' 84.

[GT03] S. Goldwasser and Y. Tauman. On the (In)security of the Fiat-Shamir Paradigm. Cryptology ePrint Archive, Report 2003/034, 2003. Extended abstract appeared in FOCS' 03.

[Had00] S. Hada. Zero-Knowledge and Code Obfuscation. In *AsiaCrypt '00*, pages 443–457, 2000. LNCS No. 1976.

[HT99] S. Hada and T. Tanaka. On the Existence of 3-Round Zero-Knowledge Protocols. Cryptology ePrint Archive, Report 1999/009, 1999. `http://eprint.iacr.org/`.

[Has03] F. Haslam, editor. *Code Names & RAF Vocabulary.* 2003. `http://www.associations.rafinfo.org.uk/code_names.htm`.

[HILL89] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999. Preliminary versions appeared in STOC' 89 and STOC' 90.

[IL89] R. Impagliazzo and M. Luby. One-way Functions are Essential for Complexity Based Cryptography (Extended Abstract). In *Proc. 30th FOCS*, pages 230–235. IEEE, 1989.

[IW97] R. Impagliazzo and A. Wigderson. $\mathbf{P} = \mathbf{BPP}$ if $\mathbf{E}$ Requires Exponential Circuits: Derandomizing the XOR Lemma. In *Proc. 29th STOC*, pages 220–229. ACM, 1997.

[KY00] J. Katz and M. Yung. Complete Characterization of Security Notions for Private-Key Encryption. In *Proc. 32th STOC*, pages 245–254. ACM, 2000.

[Kil92] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proc. 24th STOC*, pages 723–732. ACM, 1992.

[Kil95] J. Kilian. Improved Efficient Arguments (Preliminary version). In *Crypto '95*, pages 311–324, 1995. LNCS No. 963.

[KP00] J. Kilian and E. Petrank. Concurrent Zero-Knowledge in Poly-logarithmic Rounds. Cryptology ePrint Archive, Report 2000/013, 2000. Extended abstract appeared in STOC' 01.

[KPR98] J. Kilian, E. Petrank, and C. Rackoff. Lower bounds for zero knowledge on the Internet. In *Proc. 39th FOCS*, pages 484–492. IEEE, 1998.

[Lam79] L. Lamport. Constructing Digital Signatures from a One-Way Function. Technical Report CSL-98, SRI International, Oct. 1979.

[Lev86] L. A. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

[Lin01] Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. In *Crypto '01*, pages 171–189, 2001. LNCS No. 2139.

[Lin03a] Y. Lindell. Bounded-concurrent secure Two-party Computation without Setup Assumptions. In *Proc. 35th STOC*. ACM, 2003.

[Lin03b] Y. Lindell. A Simpler Construction of CCA2-Secure Public-Key Encryption Under General Assumptions. pages 241–254, 2003. LNCS No. 2656.

[LR86] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988. Preliminary version in STOC' 86.

[LFKN90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proc. 31st FOCS*, pages 2–10. IEEE, 1990.

[Mer89] R. C. Merkle. A certified digital signature. In *Crypto '89*, pages 218–238, 1989. LNCS No. 435.

[Mic94] S. Micali. CS proofs. In *Proc. 35th FOCS*, pages 436–453. IEEE, 1994.

[NSS99] D. Naccache, A. Shamir, and J. P. Stern. How to Copyright a Function? pages 188–196, 1999. LNCS No. 1560.

[Nao89] M. Naor. Bit Commitment Using Pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991. Preliminary version in CRYPTO' 89.

[NR97] M. Naor and O. Reingold. Number-theoretic Constructions of Efficient Pseudo-random Functions. In *Proc. 38th FOCS*, pages 458–467. IEEE, 1997.

[Nat95] National Institute of Standards and Technology. *FIPS PUB 180-1: Secure Hash Standard*. National Institute for Standards and Technology, Apr. 1995.

[Net03] Network Associates Inc. $1.8 Million Awarded to Network Associates Laboratories to Develop New Technologies for Protection of Critical Software, 2003. Press statement available from Network Associate's website on `http://www.networkassociates.com/us/about/press/corporate/2003/20030121%.htm` .

[NW88] N. Nisan and A. Wigderson. Hardness vs Randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, Oct. 1994. Preliminary version in FOCS' 88.

[Par90] E. Partridge. *Dictionary of RAF Slang*. Pavilion Books, 1990. First edition printed in 1945.

[PRS92] M. Prabhakaran, A. Rosen, and A. Sahai. Concurrent Zero Knowledge with Logarithmic Round-Complexity. In *Proc. 33rd FOCS*. IEEE, 1992.

[Rab79] M. O. Rabin. Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology, Jan. 1979.

[RK99] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *Eurocrypt '99*, 1999. LNCS No. 1592.

[RAD78] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of secure computation (Workshop, Georgia Inst. Tech., Atlanta, Ga., 1977)*, pages 169–179. Academic, New York, 1978.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb 1978.

[Ros00] A. Rosen. A Note on the Round-Complexity of Concurrent Zero-Knowledge. In *Crypto '00*, 2000. LNCS No. 1880.

[Sah99] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proc. 40th FOCS*, pages 543–553. IEEE, 1999.

[SYY99] T. Sander, A. Young, and M. Yung. Non-interactive Cryptocomputing for NC^1. In *Proc. 40th FOCS*, pages 554–566. IEEE, 1999.

[Sch01] P. Scholten. An Historical Prespective: Albert Abrams, The Physician Who Made Millions Out of Electricity. *San Francisco Medicine*, 74(5), 2001. Available on `http://www.sfms.org/sfm/sfm501k.htm`.

[TW87] M. Tompa and H. Woll. Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information. In *Proc. 28th FOCS*, pages 472–482. IEEE, 1987.

[vD98] F. van Dorsselaer. Obsolescent Feature. Winning entry for the *1998 International Obfuscated C Code Contest*, 1998. `http://www.ioccc.org/`.

[VO03] P. C. Van Oorschot. Revisiting Software Protection. In *6th International Information Security Conference (ISC 2003)*, pages 1–13, 2003. Springer LNCS 2851. Available from `http://www.scs.carleton.ca/~paulv/` .

[Vas96] G. Vassilatos. *"Nicola Tesla's Black-Box": excerpt from "Secrets of Cold War Technology"*, volume Adventures Unlimited, pages 86–93. 1996. See `http://www.frank.germano.com/blackbox.htm`.

[Vle99] R. V. Vleck. The Electronic Reactions of Albert Abrams. *American Artifacts*, 39, 1999. `http://www.americanartifacts.com/smma/abrams/abrams.htm`.

[Wag00] D. Wagner. Declaration of David Wagner In Opposition to Order to Show Cause, 2000. Available on `http://www.eff.org/IP/Video/DVDCCA_case/20000107-pi-motion-wagnerdec.ht%ml` .

[Wil03] D. Wilton. Word Origins Web-Site: Letter B, 2003. `http://www.wordorigins.org/wordorb.htm`.